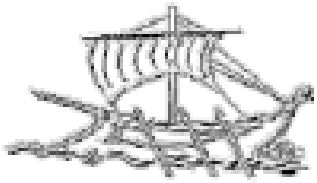


ΤΕΙ ΠΕΙΡΑΙΑ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. "ΕΦΑΡΜΟΣΜΕΝΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη web εφαρμογών με χρήση Apache Wicket

Δημήτριος Α. Σκαρπέτης

Εισηγητής: Νικόλαος Ζ. Ζάχαρης, Καθηγητής

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2017

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη web εφαρμογών με χρήση Apache Wicket

Δημήτριος Α. Σκαρπέτης
A.M. ais0067

Εισηγητής: Νικόλαος Ζ. Ζάχαρης, Καθηγητής

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Σκαρπέτης Δημήτριος, του Ανδρέα, με αριθμό μητρώου ais0067 φοιτητή του Π.Μ.Σ Εφαρμοσμένα Πληροφοριακά Συστήματα του Τμήματος Μηχανικών Η/Υ Συστημάτων Τ.Ε. του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Διπλωματικής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Διπλωματική Εργασία (Δ.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Δ.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Δίπλωμα, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Δ.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Δ.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά θα ήθελα να ευχαριστήσω την οικογένειά μου για την ηθική και πρακτική στήριξη που μου έδειξαν καθ' όλη τη διάρκεια των σπουδών μου.

Θα ήθελα να ευχαριστήσω εκ' βαθέων τον καθηγητή κ. Νικόλαο Ζάχαρη για την ανάθεση του θέματος, τις πολύτιμες συμβουλές του, το ενδιαφέρον του αλλά και τον χρόνο που διέθεσε για την περάτωση της παρούσας διπλωματικής εργασίας. Επίσης θα ήθελα να ευχαριστήσω, όλους του καθηγητές του μεταπτυχιακού για τις πολύτιμες γνώσεις που μου προσέφεραν.

Τέλος, ένα μεγάλο ευχαριστώ στους φίλους και συμφοιτητές μου για τη συνεργασία που είχαμε αυτά τα δύο χρόνια.

ΠΕΡΙΛΗΨΗ

Στόχος της παρούσας διπλωματικής είναι να γίνουν γνωστά στον αναγνώστη η δομή, η φιλοσοφία και οι λειτουργίες που διέπουν το εργαλείο κατασκευής διαδικτυακών εφαρμογών Apache Wicket. Στο πρώτο κεφάλαιο επιχειρείται μια σύντομη παρουσίαση των δημοφιλέστερων εργαλείων κατασκευής διαδικτυακών εφαρμογών της αγοράς. Στα επόμενα δυο κεφάλαια γίνεται μια εκτενής ανάλυση της αρχιτεκτονικής και των χαρακτηριστικών του Apache Wicket, μέσα από σύντομα και απλά παραδείγματα κώδικα που χρησιμοποιήθηκαν στην κατασκευή της ηλεκτρονικής βιβλιοθήκης. Στη συνέχεια γίνεται αναφορά στις προδιαγραφές και στη υπόλοιπες τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Τέλος παρουσιάζονται οι λειτουργίες που μπορούν να πραγματοποιήσουν, κατά τη διάρκεια της περιήγησής τους στη διαδικτυακή βιβλιοθήκη, οι εγγεγραμμένοι χρήστες και ο διαχειριστής καθώς και τα συμπεράσματα που προέκυψαν κατά την διάρκεια. εκπόνησης της εργασίας.

ABSTRACT

The target of this thesis is the structure, philosophy and functions that related to the web framework Apache Wicket, to be known by the reader. In the first chapter there is an attempt of a brief presentation of the most popular web frameworks on the Market. In the next two chapters there is an extensive analysis of the architecture and characteristics of Apache Wicket, through short and simple code examples used in the development of the electronic library application. Then are presented the specifications and the rest technologies that have been used in the development of the application. In the last chapter are presented the features that can be realized by the registered users and the administrator that can be used during their browsing at the electronic library and the conclusions that occurred during thesis preparation.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Διαδικτυακές εφαρμογές

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Apache Wicket, Web framework, Java, Hibernate, JPA, ORM, Guice, postgreSQL, BootStrap, JUnit.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ

Κεφάλαιο 1
Web Framework

1.1 Τι είναι web εφαρμογή

Με τον όρο εφαρμογή, χαρακτηρίζουμε το λογισμικό το οποίο εγκαθίσταται σε έναν ηλεκτρονικό υπολογιστή και έχει σχεδιαστεί ώστε να πραγματοποιεί συγκεκριμένες λειτουργίες, να επιτυγχάνει συγκεκριμένους στόχους και να εξάγει στον χρήστη την επιθυμητή πληροφορία.

Ο όρος web εφαρμογή (web application) χαρακτηρίζει κάθε εφαρμογή η οποία είναι διαθέσιμη στου χρήστες μέσω του διαδικτύου και ο χρήστης χρειάζεται μόνο τον περιηγητή του (web browser) για να την χρησιμοποιήσει.

Οι web εφαρμογές εκτελούνται σε εξυπηρετητές (server) και παρέχουν τις υπηρεσίες τους σε περισσότερους από έναν χρήστη.

1.2 Πλεονεκτήματα web εφαρμογών

- Είναι προσβάσιμες από οποιαδήποτε ηλεκτρονική συσκευή διαθέτει internet χωρίς την εγκατάσταση επιπλέον λογισμικού, παρά μόνο με τον περιηγητή ο οποίος είναι προ εγκατεστημένος σε όλα τα λειτουργικά συστήματα.
- Είναι συμβατές με όλα τα λειτουργικά συστήματα (π.χ. windows, mac, linux κ.τ.λ.). Εφόσον η εφαρμογή εκτελείται μέσω του περιηγητή του διαδικτύου και όχι στον υπολογιστή του χρήστη, την καθιστά ικανή να εκτελείται σε όλα τα λειτουργικά συστήματα.
- Από τη στιγμή που οι διαδικτυακές εφαρμογές δεν εκτελούνται στην ηλεκτρονική συσκευή του χρήστη, δεν καταναλώνουν πόρους από το υπολογιστικό του σύστημα.
- Οι web εφαρμογές δεν καταλαμβάνουν χώρο στο σκληρό δίσκο του χρήστη αφού το σύνολο της εφαρμογής είναι αποθηκευμένο στον εξυπηρετητή.
- Σε μία τοπική εφαρμογή η αναβάθμιση θα πρέπει να γίνετε ξεχωριστά σε κάθε υπολογιστή. Αντίθετα σε μια διαδικτυακή εφαρμογή η αναβάθμιση πραγματοποιείται μόνο στον εξυπηρετητή που φιλοξενεί την εφαρμογή και ταυτόχρονα το αναβαθμισμένο πρόγραμμα είναι διαθέσιμο σε όλους τους χρήστες.

1.3 Μειονεκτήματα web εφαρμογών

- Μη δυνατότητα χρήσης της εφαρμογής χωρίς σύνδεση στο διαδίκτυο, σε αντίθεση με μια εφαρμογή που είναι εγκατεστημένη στο σκληρό δίσκο του χρήστη.
- Ένα ακόμα μειονέκτημα αφορά την ασυμβατότητα των περιηγητών με την συγκεκριμένη έκδοση της εφαρμογής. Σε περίπτωση που δεν έχει προβλεφθεί η λειτουργία κάποιου χαρακτηριστικού της εφαρμογής σε συγκεκριμένο περιηγητή, αυτό μπορεί να δημιουργήσει προβλήματα στην εφαρμογή με αποτέλεσμα να μην λειτουργεί σωστά. Για να αποφευχθούν αυτά τα προβλήματα πρέπει ο προγραμματιστής κατά την κατασκευή της εφαρμογής να έχει προβλέψει για την ομαλή λειτουργία της, τουλάχιστον στους πιο δημοφιλείς browser.
- Υπάρχει μεγαλύτερος κίνδυνος από πλευράς ασφάλειας για την διαρροή πληροφοριών. Για την ανάπτυξη μιας ασφαλούς διαδικτυακής εφαρμογής, είναι απαραίτητη η θωράκιση σε επίπεδο δικτύου, υπολογιστικού συστήματος και εφαρμογής.

1.4 Τι είναι web framework

Το Web framework ή web application framework είναι ένα σύνολο εργαλείων τα οποία χρησιμεύουν για την κατασκευή και συντήρηση διαδικτυακών εφαρμογών (web applications), διαδικτυακών υπηρεσιών (web services) και ιστοσελίδων (websites).

Μερικές φορές θα πρέπει να αναρωτηθούμε, γιατί ακολουθούμε καθημερινά την ίδια διαδρομή με το αυτοκίνητο μας, αντί να χρησιμοποιήσουμε τη νεόκτιστη γέφυρα η οποία θα συντόμευε και θα έκανε ευκολότερη τη διαδρομή μας. Μπορούμε να θεωρήσουμε ένα web framework ως γέφυρα η οποία θα μας βοηθήσει να σχεδιάσουμε ταχύτερα και ευκολότερα μια εφαρμογή.

Με τη χρήση ενός web framework μπορεί να επιτευχθεί υψηλή ποιότητα λογισμικού που συνεπάγεται ένα λογισμικό πιο αποτελεσματικό (efficient), επαναχρησιμοποιήσιμο (reusable), επεκτάσιμο (extendible) και συντηρήσιμο (maintainable).

1.5 Διαχωρισμός web frameworks με βάση την αρχιτεκτονική τους

Σήμερα στο διαδίκτυο είναι διαθέσιμα πολλά frameworks τα οποία θα προσπαθήσουμε στις επόμενες παραγράφους να τα κατηγοριοποιήσουμε με βάση την αρχιτεκτονική τους.

- **Request based framework**: Το Request based framework χειρίζεται άμεσα τα εισερχόμενα αιτήματα, ο χρήστης στέλνει ένα αίτημα και ο controller με τη σειρά του την απόκριση (response) στον χρήστη. Τα Sping MVC [https://spring.io/], Struts [https://struts.apache.org/], Ruby on Rails [http://rubyonrails.org/] και Grails [https://grails.org/] είναι κάποια από τα πιο δημοφιλή request based framework.
- **Component based framework**: Μέχρι τη δημιουργία των component based frameworks το λογισμικό εξακολουθούσε να είναι όλο και πιο σύνθετο, πολύπλοκο και ακριβό. Για να ξεπεραστεί αυτή η αδυναμία του λογισμικού, τα component based εργαλεία, έδωσαν τη δυνατότητα σε κάθε νέο λογισμικό να αναπτύσσεται μέσω μικρών κομματιών κώδικα τα οποία θα μπορούν εύκολα να επαναχρησιμοποιηθούν σε διαφορετικά έργα. Γνωστά component based framework είναι τα Apache Wicket [https://wicket.apache.org/], JSF [http://www.jsf.com/], Tapestry [http://tapestry.apache.org/] και RIFE [http://rifers.org/].
- **Hybrid framework:** Ο συνδυασμός request based και component based οδήγησε στην δημιουργία των hybrid frameworks. Αντί ο controller να χειρίζεται απευθείας τις αιτήσεις (requests), το υβριδικό framework παρέχει ένα component μοντέλο το οποίο έχει την ίδια συμπεριφορά σε διαφορετικές καταστάσεις. Ένα framework που υλοποιεί την hybrid αρχιτεκτονική είναι το Onsen UI [https://onsen.io/].
- **Meta framework:** Το meta framework αποτελείται από διεπαφές (interfaces) οι οποίες παρέχουν ένα σύνολο από υπηρεσίες (services). Στο συγκεκριμένο framework μπορούν να ενσωματωθούν άλλα frameworks και components τα οποία με την υλοποίηση των διεπαφών μπορούν να επεκτείνουν τις λειτουργίες τους. Το Keel [http://www.keelframework.org/] framework υλοποιεί την αρχιτεκτονική meta.
- **RIA based framework**: Οι εφαρμογές που υλοποιούνται με το RIA (Rich Internet Application) framework [https://en.wikipedia.org/ria] έχουν πολλά

κοινά χαρακτηριστικά με τα λογισμικά που εγκαθίστανται στον τοπικό υπολογιστή (desktop application software). Το RIA framework χρησιμοποιεί ένα client side container για να ελαχιστοποιήσει το συνολικό αριθμό των αιτήσεων στον server. Κάθε φορά που ο χρήστης κάνει κλικ σε ένα link ή πατάει κάποιο κουμπί, αντί να φορτώνεται ολόκληρη η σελίδα το framework χειρίζεται τοπικά τις αιτήσεις. Παράδειγμα RIA based framework είναι το Flex [http://flex.apache.org/].

1.6 Πλεονεκτήματα web frameworks

- **Αποδοτικότητα** (Efficiency): Εργασίες οι οποίες συνήθως χρειάζονται αρκετές ώρες και εκατοντάδες γραμμές κώδικα, μπορούν να υλοποιηθούν μέσα σε λίγα λεπτά με τη χρήση ενσωματωμένων διαδικασιών που παρέχονται από το framework. Η δουλειά του προγραμματιστή γίνετε πιο εύκολη, πιο γρήγορη και κατά συνέπεια περισσότερο αποτελεσματική.
- **Ασφάλεια** (Security): Τα δημοφιλέστερα web frameworks παρέχουν πολλές υλοποιήσεις που εξασφαλίζουν την ασφάλεια της εφαρμογής. Χαρακτηριστικά ασφάλειας όπως είναι η πιστοποίηση χρηστών και των δικαιωμάτων τους, διαχειρίζονται από το framework.
- **Υποστήριξη από την κοινότητα** (Community): Το μεγάλο πλεονέκτημα είναι η υποστήριξη της κοινότητας που υπάρχει πίσω από κάθε framework. Αν ο σχεδιαστής εντοπίσει κάποια ευπάθεια ή οποιοδήποτε άλλο πρόβλημα στην εφαρμογή, τότε μπορεί να απευθυνθεί άμεσα στη κοινότητα ώστε να του προταθούν λύσεις. Το ίδιο πρόβλημα μπορεί να το έχει αντιμετωπίσει κάποιος άλλος στο παρελθόν και η λύση να είναι ήδη δημοσιευμένη στο forum της κοινότητας.
- **Κόστος** (Cost): Τα πιο δημοφιλή web framework είναι εντελώς δωρεάν. Το κόστος για τον τελικό πελάτη είναι μικρότερο, αν αναλογιστούμε ότι το framework παρέχει πολλές “έτοιμες” διαδικασίες στους προγραμματιστές, με συνέπεια να υλοποιούνται γρηγορότερα διάφορες λειτουργίες.

1.7 Μειονεκτήματα web frameworks

- **Μαθαίνεις το framework, όχι τη γλώσσα:** Αυτό είναι και το μεγαλύτερο πρόβλημα. Αν χρησιμοποιείς ένα framework και δεν γνωρίζεις σε υψηλό επίπεδο τη γλώσσα προγραμματισμού που χρησιμοποιεί το συγκεκριμένο

εργαλείο, ο κίνδυνος είναι ότι θα μάθεις να χειρίζεσαι το framework και όχι τη γλώσσα. Ο τρόπος που γράφεις κώδικα σε jQuery είναι διαφορετικός από ότι σε JavaScript. Το ότι γνωρίζει κάποιος jQuery δεν σημαίνει απαραίτητα ότι γνωρίζει και JavaScript.

- **Περιορισμός (Restriction):** Ένα ακόμα μειονέκτημα είναι ότι δεν μπορούν να αλλαχθούν οι κεντρικές λειτουργίες του framework, πράγμα που σημαίνει ότι όταν χρησιμοποιείς ένα τέτοιο εργαλείο είσαι αναγκασμένος να δουλεύεις μέσα σε συγκεκριμένα πλαίσια και όρια. Γι' αυτό το λόγο, πριν ξεκινήσουμε την υλοποίηση μιας web εφαρμογής, πρέπει να έχουμε επιλέξει προσεχτικά το web framework το οποίο ταιριάζει στις ανάγκες μας.
- **Δημόσιος κώδικας:** Από τι στιγμή που ο κώδικας είναι διαθέσιμος σε όλους, σημαίνει ότι είναι διαθέσιμος και σε ανθρώπους με κακές προθέσεις. Τέτοιοι άνθρωποι μπορούν να μελετήσουν των κώδικα της εφαρμογής σας, να βρουν τυχόν ελαττώματα και να τα χρησιμοποιήσουν εναντίον σας.

1.8 Δημοφιλέστερα web frameworks ανά γλώσσα προγραμματισμού

Τη τελευταία δεκαπενταετία υπήρξαμε μάρτυρες μιας δραματικής αλλαγής στο τρόπο ανάπτυξης μεγάλων και επαγγελματικών εφαρμογών. Η χρήση βαρέων αρχιτεκτονικών (Heavyweight Architectures), όπως είναι τα Enterprise Java Beans, έχει περιοριστεί σημαντικά και τη θέση τους έχουν πάρει πιο ελαφριά web frameworks. Διάφορες λειτουργίες που μέχρι τότε ήταν εξαρτώμενες από άλλες τεχνολογίες, όπως η σχεσιακή αντιστοίχιση των αντικειμένων (Object Relational Mapping) αλλά και τα συστήματα διαχείρισης συναλλαγών (transaction management systems), έχουν αντικατασταθεί από πιο απλές και εναλλακτικές. Παρακάτω θα γίνει αναφορά των πιο δημοφιλών web framework ανά γλώσσα προγραμματισμού.

1.8.1 Java web frameworks

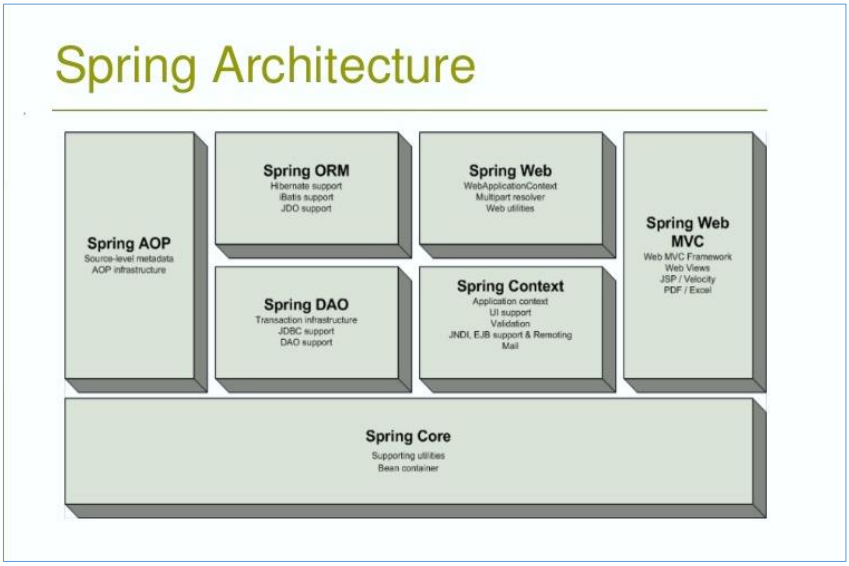
Spring MVC

Το Spring [https://spring.io/] είναι ένα ελεύθερο περιβάλλον εργασίας για εφαρμογές Java που δημιουργήθηκε από τον Rob Johnson. Αναπτύχθηκε για να αντιμετωπίσει την πολυπλοκότητα ανάπτυξης επαγγελματικών εφαρμογών. Με την χρήση μικρών και απλών σε κώδικα JavaBeans, το Spring framework κάνει εφικτό την υλοποίηση πραγμάτων που προηγουμένως μπορούσαν να γίνουν μόνο μέσω enterprise JavaBeans. Το Spring δεν είναι μόνο χρήσιμο για την ανάπτυξη server-

side εφαρμογών, αλλά μπορεί να βοηθήσει στην απλοποίηση του κώδικα, τον ευκολότερο και πιο αποτελεσματικό έλεγχο και τη χαλαρή διασύνδεση (loose coupling) κάθε Java εφαρμογής. Το Spring είναι ένας ελαφρύς (lightweight) aspect oriented container ο οποίος εφαρμόζει dependency injection, ενώ παράλληλα αποτελεί και περιβάλλον εργασίας (εικόνα 1.1).

- **Lightweight**: Με τον όρο αυτό αναφερόμαστε τόσο στο μέγεθος όσο και στον επιπλέον φόρτο (overhead). Το Spring Framework μπορεί να συμπεριληφθεί σε ένα απλό “jar” αρχείο λίγων megabytes, ενώ ο επιπρόσθετος φόρτος εργασίας που απαιτείται είναι μηδαμινός. Ο προγραμματιστής χρειάζεται να κάνει ελάχιστες αλλαγές στον κώδικα της εφαρμογής που αναπτύσσει, έτσι ώστε να επωφεληθεί από τον πυρήνα της.
- **Dependency Injection:** Η Spring προάγει τη χαλαρή διασύνδεση χρησιμοποιώντας μία τεχνική που είναι γνωστή ως dependency injection. Όταν εφαρμόζεται η DI, τα αντικείμενα λαμβάνουν παθητικά τις εξαρτήσεις τους (dependencies) αντί να τις δημιουργούν ή να τις αναζητούν μόνα τους. Είναι κάτι σαν ένα αντίστροφο “Java Naming and Directory Interface”, αντί ένα αντικείμενο να ψάχνει μόνο του για τις εξαρτήσεις του σε έναν container, ο ίδιος ο container δίνει τις εξαρτήσεις στο αντικείμενο χωρίς να περιμένει πρώτα να ερωτηθεί.
- **Aspect Oriented Programming:** Το Spring υποστηρίζει Aspect Oriented Programming. Αυτό έχει σαν αποτέλεσμα τη δημιουργία εφαρμογών με περισσότερη συνοχή ακόμη και αν είναι απαραίτητη η συνύπαρξη διαφορετικών λογικών λειτουργίας. Κάθε αντικείμενο της εφαρμογής κάνει μόνο ό,τι το αφορά και δεν είναι υπεύθυνο για λειτουργίες που έχουν να κάνουν με άλλα συστήματα. Τέτοιες λειτουργίες μπορεί να είναι η υποστήριξη συνδιαλλαγών και η καταγραφή των διαφόρων κινήσεων (logging).
- **Container:** Το Spring framework αποτελεί έναν container με την έννοια ότι διαχειρίζεται τον κύκλο ζωής και τη διαμόρφωση των αντικειμένων της εφαρμογής. Ο προγραμματιστής μπορεί να ορίσει πως θέλει να δημιουργούνται τα αντικείμενα, καθώς και ποιες να είναι οι μεταξύ τους σχέσεις.

- **Framework:** Το Spring δίνει τη δυνατότητα να δημιουργηθούν πολύπλοκες εφαρμογές με το συνδυασμό άλλων, λιγότερο πολύπλοκων, κομματιών. Στο Spring τα αντικείμενα δημιουργούνται σε απλά XML αρχεία. Επιπλέον παρέχει πολλές δομικές λειτουργίες, επιτρέποντας στον προγραμματιστή να ασχοληθεί περισσότερο με τη λογική της εφαρμογής του.



Εικόνα 1.1: Αρχιτεκτονική Spring MVC [https://docs.spring.io/architecture].

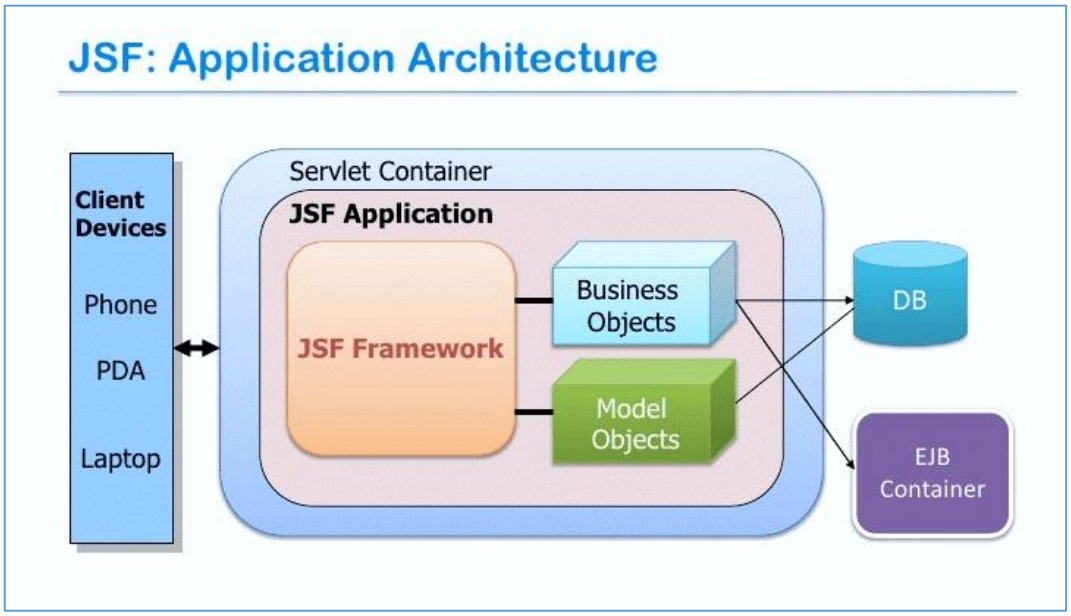
JSF

Το JSF [http://www.jsf.com/] είναι ένα framework ανοιχτού κώδικα για την ανάπτυξη Java web εφαρμογών, το οποίο στηρίζεται στην απλοποίηση της σχεδίασης σελίδων και στην ευκολότερη διαχείριση αυτών. Αποτελείται από σύνολα εργαλείων διεπαφής χρήστη (components) και υποστηρίζει ένα μοντέλο προγραμματισμού προσανατολισμένο σε γεγονότα (event driven).

Οι εφαρμογές σε JSF χρησιμοποιούν το συγκεκριμένο framework για να υλοποιήσουν το Servlet API. Χρησιμοποιούν το πρωτόκολλο HTTP μέσω του Servlet API και κάποιο είδος τυπικής τεχνολογίας εμφάνισης, όπως οι Java Server Pages, η οποία καθορίζει την παρουσίαση των σελίδων.

Οι JSF εφαρμογές έχουν τα UI (backing beans) αντικείμενα, τα οποία και μπορούν να κρατούν αυτόματα συγχρονισμένα με τα αντικείμενα της Java που συλλέγουν τις τιμές εισαγωγής των χρηστών και αποκρίνονται στα γεγονότα που εμπλέκονται στη λογική της εφαρμογής. Έτσι, τα JSF χρησιμοποιούν το Servlet API για την υποδομή τους, αλλά επιτρέπουν την εργασία σε ένα πιο υψηλό επίπεδο αφαίρεσης, δίνοντας τη δυνατότητα ανάπτυξης web εφαρμογών χωρίς να υπάρχει

ανησυχία για το HTTP ή τις λεπτομέρειες του ίδιου του Servlet API. Επιτρέπουν στους προγραμματιστές να σκέφτονται και να λειτουργούν με αντικειμενοστραφείς όρους αντί για όρους αιτημάτων, αποκρίσεων, και markup καλύπτοντας έτσι τις πολυπλοκότητες της web ανάπτυξης. Τα JSF, έχουν σαν στόχο να καταστήσουν την ανάπτυξη των Web εφαρμογών γρηγορότερη και ευκολότερη (εικόνα 1.2).



Εικόνα 1.2: Αρχιτεκτονική JSF [https://www.slideshare.net/architecture].

Apache Wicket

Το Wicket [https://wicket.apache.org/] είναι ένα framework που χρησιμοποιεί τη Java γλώσσα προγραμματισμού και ανήκει στην μεγάλη οικογένεια της Apache Foundation. Το ιδιαίτερο χαρακτηριστικό που έχει είναι ότι διαχωρίζει την εμφάνιση από την λογική. Αυτό επιτυγχάνεται με το να απαιτείται για κάθε τι που εμφανίζεται, αφενός ένα στοιχείο HTML σε κάποια σελίδα και αφετέρου αντίστοιχο Wicket component στιγμιότυπο που να δημιουργείται στον κατασκευαστή κάποιας ομώνυμης με την σελίδα κλάσης. Επίσης διαθέτει δυνατότητα χρήσης AJAX [https://en.wikipedia.org/wiki/Ajax] τεχνολογίας αλλά παράλληλα και κλασικού submit, refresh. Αναλυτικότερα θα παρουσιαστεί το Apache Wicket στα επόμενα κεφάλαια.

1.8.2 PHP web framework

Laravel

Το Laravel [https://laravel.com/] είναι ανοικτού κώδικα PHP web framework που δημιουργήθηκε από τον Taylor Otwell το 2011, με κύριο στόχο την δημιουργία

διαδικτυακών εφαρμογών. Η αρχιτεκτονική του Laravel βασίζεται στο Model View Controller, ενώ αυτό που το κάνει να ξεχωρίζει από τα υπόλοιπα framework είναι το ευκολονόητο συντακτικό. Σύμφωνα με τον δημιουργό του, το Laravel θέλει να κάνει το προγραμματισμό διασκεδαστικό και για αυτό το λόγο δημιουργήθηκε ώστε να πετυχαίνει τους παρακάτω στόχους:

- **Απλό**: Οι λειτουργίες του Laravel είναι απλές στην κατανόηση και στην υλοποίηση. Αν κάποιος έχει δουλέψει στο παρελθόν με κάποιο άλλο MVC framework τότε η μετάβαση στο Laravel θα είναι εύκολη.
- **Κομψό**: Το Laravel βασίζεται στα τρέχοντα πρότυπα της βιομηχανίας που έχει ως αποτέλεσμα να μην απαιτούνται περίπλοκοι κώδικες για να εκτελεστούν λειτουργίες που κανονικά είναι απλές. Σε γενικές γραμμές το Laravel χρειάζεται ελάχιστη παραμετροποίηση για να ρυθμιστεί και να δουλέψει.
- **Τεκμηρίωση**: Η τεκμηρίωση του Laravel είναι πλήρης και πάντα ενημερωμένη. Εξίσου σημαντικό για κάποιον που θέλει να μάθει Laravel είναι ότι υπάρχουν πάρα πολλές πηγές με οδηγούς και συμβουλές στο διαδίκτυο.

1.8.3 .NET web framework

ASP .NET MVC

Το ASP.NET [https://www.asp.net/mvc] είναι ένα εργαλείο για διαδικτυακές εφαρμογές που έχει αναπτυχθεί και διατίθεται στο εμπόριο από τη Microsoft για να επιτρέπει στους προγραμματιστές την δημιουργία δυναμικών ιστοσελίδων, διαδικτυακών εφαρμογών και web υπηρεσιών. Αποτελεί μέρος της πλατφόρμας .NET και είναι διάδοχος της τεχνολογίας ASP. Η ASP.NET είναι ενσωματωμένη στο Common Language Runtime, επιτρέποντας στους προγραμματιστές να γράφουν κώδικα ASP.NET χρησιμοποιώντας οποιαδήποτε υποστηριζόμενη .NET γλώσσα. Τα κυριότερα χαρακτηριστικά του ASP.NET MVC είναι:

- **Σελίδες**: Οι ASP.NET ιστοσελίδες είναι το κύριο συστατικό για την ανάπτυξη εφαρμογών. Οι ιστοσελίδες περιέχουν στατικές (X)HTML σημάνσεις, καθώς και σήμανση για server side controls, όπου οι προγραμματιστές τοποθετούν τον απαιτούμενο στατικό και δυναμικό περιεχόμενο για την ιστοσελίδα. Οι φόρμες είναι αρχεία με κατάληξη “aspx”.

- **Προσαρμοσμένα στοιχεία ελέγχου:** Ο προγραμματιστής μπορεί να δημιουργήσει προσαρμοσμένα στοιχεία ελέγχου για τις web εφαρμογές, όπου όλος ο κώδικας είναι συγκεντρωμένος σε μια DLL βιβλιοθήκη.
- **Κατάσταση Συνόδου:** Η σύνοδος αποτελείται από μεταβλητές χρήστη που είναι ενεργές κατά τη διάρκεια της συνόδου του χρήστη. Οι μεταβλητές αυτές προσεγγίζονται με τη λογική των session και είναι μοναδικές για κάθε σύνοδο του χρήστη. Το ASP.NET framework υποστηρίζει τρεις διαφορετικές καταστάσεις διατήρησης server side μεταβλητών συνόδου.
    - **In Process λειτουργία:** Οι μεταβλητές διατηρούνται στο πλαίσιο της διεργασίας ASP.NET. Σε αυτή τη λειτουργία οι μεταβλητές καταστρέφονται όταν η διαδικασία ASP.NET ανακυκλώνεται ή αν κλείσει.
    - **ASP State λειτουργία:** Η ASP State έχει σαν αρμοδιότητα να διατηρεί τις μεταβλητές κατάστασης. Αυτή η λειτουργία επιτρέπει σε μια εφαρμογή του ASP.NET να γίνεται ισορροπημένα και να διαμοιράζεται σε πολλούς servers.
    - **SqlServer λειτουργία:** Οι μεταβλητές αποθηκεύονται σε μια βάση δεδομένων, επιτρέποντάς τους να συνεχίζουν να υφίστανται ακόμη και μετά την παύση λειτουργίας της ASP.NET διαδικασίας.

1.8.4 Ruby web framework

Ruby On Rails

Το Ruby On Rails [http://rubyonrails.org/] είναι ένα πολύ διαδεδομένο web framework γραμμένο σε Ruby. Δημιουργήθηκε από τον David Heinemeier Hansson τον Ιούλιο του 2004 και από τότε αυξάνεται όλο και περισσότερο η χρήση του για κατασκευές διαδικτυακών εφαρμογών. Το Rails βασίζεται στο πρότυπο σχεδίασης Model View Controller σύμφωνα με το οποίο γίνεται διαχωρισμός του κώδικα σε τρία διαφορετικά κομμάτια για περισσότερη ευκολία συγγραφής. Το MVC στο Rails υλοποιείται βάση τριών βιβλιοθηκών:

- **ActionView:** Το ActionView αναλαμβάνει την εμφάνιση των δεδομένων της εφαρμογής στο χρήστη. Όταν ο ελεγκτής δεχθεί ένα αίτημά, μέσω του ActionView, καλείται το αντίστοιχο template της μεθόδου. Οι προβολές εμφανίζουν τα δεδομένα που είναι διαθέσιμα μέσω του ελεγκτή.

Με αυτό τον τρόπο γίνεται διαχωρισμός του τρόπου διαχείρισης των δεδομένων καθώς και του τρόπου εμφάνισής τους.

- **ActionController**: Στο Ruby on Rails το ActionController είναι υπεύθυνο στο να παραλαμβάνει τις αιτήσεις από το χρήστη και να κάνει τις αντίστοιχες ενέργειες οι οποίες είναι είτε σύνδεση με τη βάση είτε εμφάνιση κώδικα HTML.
- **ActiveRecord:** Το ActiveRecord αντιπροσωπεύει το Model. Δημιουργεί μια διασύνδεση αντικειμένου με μια σχέση της βάσης (object relational mapping). Για να βρεθεί κάτι μέσα σε αυτή, αντί να ψάξει σε ολόκληρη τη βάση μέχρι να βρεθεί το στοιχείο που χρειάζεται η εφαρμογή, διαβάζεται κατευθείαν η στήλη του πίνακα του αντικειμένου που χρησιμοποιείται στο πρόγραμμα.

1.8.5 Python web framework

Django

Το Django [https://www.djangoproject.com/] είναι ένα εργαλείο ανάπτυξης κώδικα σχεδιασμένο με πρωταρχικό στόχο να διευκολύνει τη δημιουργία διαδικτυακών εφαρμογών. Το Django έχει σχεδιαστεί ακολουθώντας την αρχιτεκτονική λογισμικού MVC, παρέχοντας δυνατότητες γρήγορης και εύκολης ολοκλήρωσης συνηθισμένων εργασιών της ανάπτυξης διαδικτυακών εφαρμογών.

Ένα επιπλέον κύριο χαρακτηριστικό της σχεδίασής του είναι ο διαχωρισμός της εφαρμογής σε εξειδικευμένα και ανεξάρτητα τμήματα που μπορούν να χρησιμοποιηθούν σε διαφορετικές εφαρμογές.

Στο πιο βασικό επίπεδο η κύρια λειτουργία που εκτελεί το Django είναι η επεξεργασία των εισερχομένων αιτημάτων και η αποστολή των κατάλληλων απαντήσεων. Η χρήση αυτόνομων μονάδων σε συνδυασμό με την χρήση της αρχιτεκτονικής MVC, δίνει τη δυνατότητα στο Django να είναι εύκολα παραμετροποιήσιμο. Οι διαδικασίες που πραγματοποιούνται εσωτερικά για την επεξεργασία των αιτημάτων και την αποστολή των απαντήσεων περιγράφονται από τον κύκλο αιτημάτων - απαντήσεων.

Το Django δέχεται ως είσοδο ένα αίτημα. Αρχικά η μονάδα URL Configuration ελέγχει αν ο υπερσύνδεσμος του αιτήματος αντιστοιχεί σε κάποια προβολή και καλεί με παράμετρο το αίτημα. Η προβολή με τη σειρά της, ανάλογα την προγραμματισμένη λειτουργία, επεξεργάζεται το αίτημα και επιστρέφει την

απάντηση. Συνήθως για τη δημιουργία της απάντησης η προβολή ανακτά ή τροποποιεί δεδομένα από τα απαραίτητα μοντέλα και τα αποθηκεύει προσωρινά σε ένα λεξιλόγιο. Τέλος θα επιστρέψει ένα έγγραφο HTML δημιουργημένο από ένα Template και το λεξιλόγιο.

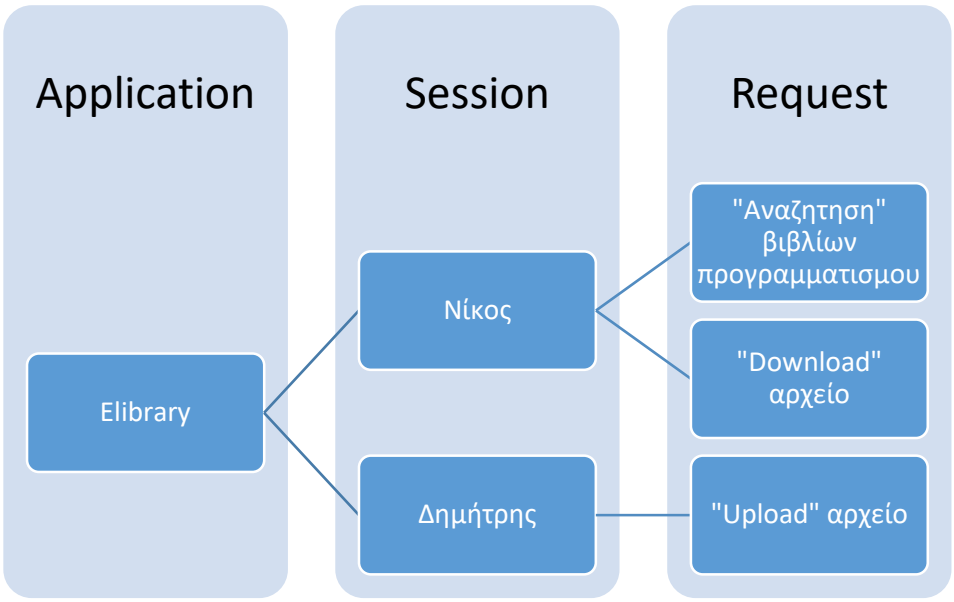
Κεφάλαιο 2
Αρχιτεκτονική του Apache Wicket

2.1 Εισαγωγή

Το Apache Wicket είναι ένα ανοικτού κώδικα, component oriented, εργαλείο κατασκευής διαδικτυακών εφαρμογών (web application framework). Δημιουργήθηκε από τον Jonathan Locke τον Απρίλιο του 2004 και η πρώτη έκδοση (version 1.0) έγινε διαθέσιμη στο κοινό από την εταιρεία Apache τον Ιούνιο του 2005. Αυτή τι στιγμή η έκδοση του Wicket βρίσκεται στην 7.4.0. Με το Wicket μπορούν να υλοποιηθούν διαδικτυακές εφαρμογές με τη χρήση απλών και επαναχρησιμοποιήσιμων component εφαρμόζοντας τις αρχές του αντικειμενοστραφούς προγραμματισμού (OO principles) της Java.

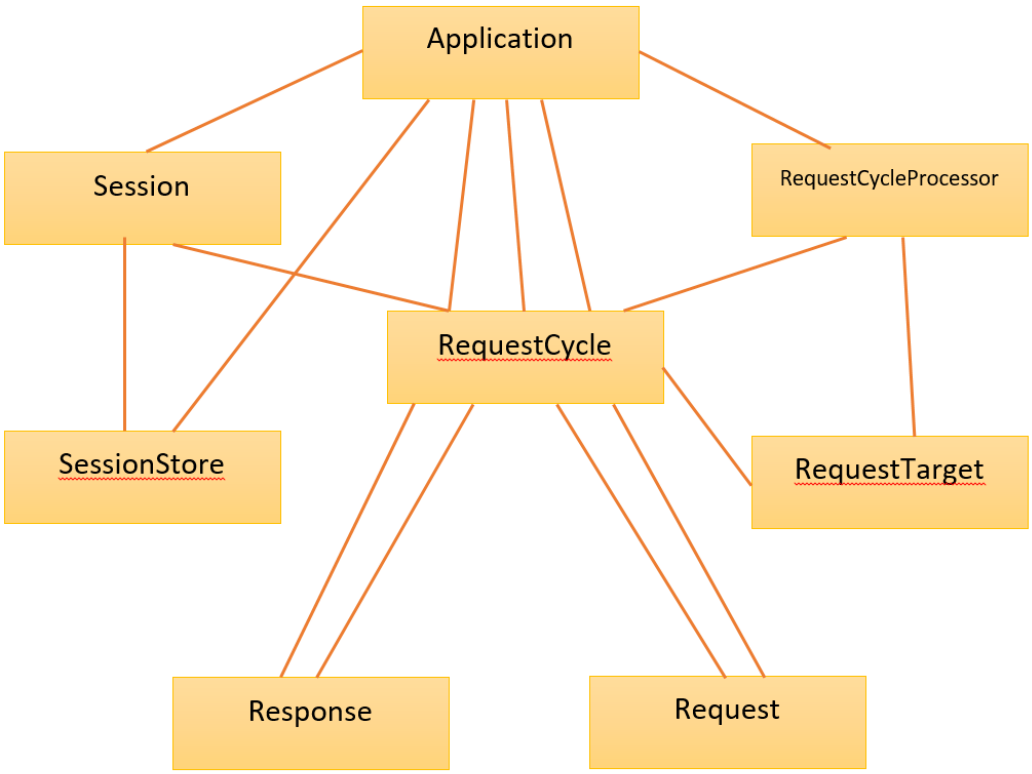
2.2 Συστατικά web εφαρμογής

Τα κυριότερα συστατικά μιας εφαρμογής, που έχει υλοποιηθεί με το εργαλείο Apache Wicket, είναι τα application, session και request. Η εφαρμογή (application) διαχειρίζεται αιτήματα (requests) από διαφορετικούς χρήστες, οι οποίοι πραγματοποιούν ενέργειες όπως αναζήτηση ή ανέβασμα (uploading) κάποιου ψηφιοποιημένου βιβλίου. Κάθε requests ενός συνδεδεμένου με την εφαρμογή χρήστη, αποτελεί μέρος του ίδιου session. Ο χρήστης εισέρχεται στην εφαρμογή, εκτελεί κάποιες ενέργειες και τέλος αποσυνδέεται από αυτή για να τερματιστεί το session (διάγραμμα 2.1).



Διάγραμμα 2.1: Ένα application, πολλαπλά sessions, τα οποία διαχειρίζονται πολλαπλά requests.

Όταν ακολουθείται η προσέγγιση του αντικειμενοστραφούς προγραμματισμού, τα πάντα υλοποιούνται μέσα από τη χρήση κλάσεων. Στο Apache Wicket εργαλείο, οι Application, Session, Request κλάσεις, η ακόμα πιο συγκεκριμένα τα στιγμιότυπα αυτών των κλάσεων παίζουν κεντρικό ρόλο στον κύκλο διεργασίας ενός αιτήματος (διάγραμμα 2.2).



Διάγραμμα 2.2: Οι κλάσεις που συμμετέχουν στη διαχείριση των request.

Application

Το Application αντικείμενο, ιεραρχικά βρίσκεται στην υψηλότερη βαθμίδα των container και συγκεντρώνει όλα τα components, HTML, properties και configuration αρχεία. Συνήθως στη κλάση αυτή δηλώνεται σαν όνομα στο πρώτο συνθετικό της, ο τίτλος της εφαρμογής. Στην περίπτωση μας, στη συγκεκριμένη κλάση έχει δοθεί το όνομα “ElibraryApplication”.

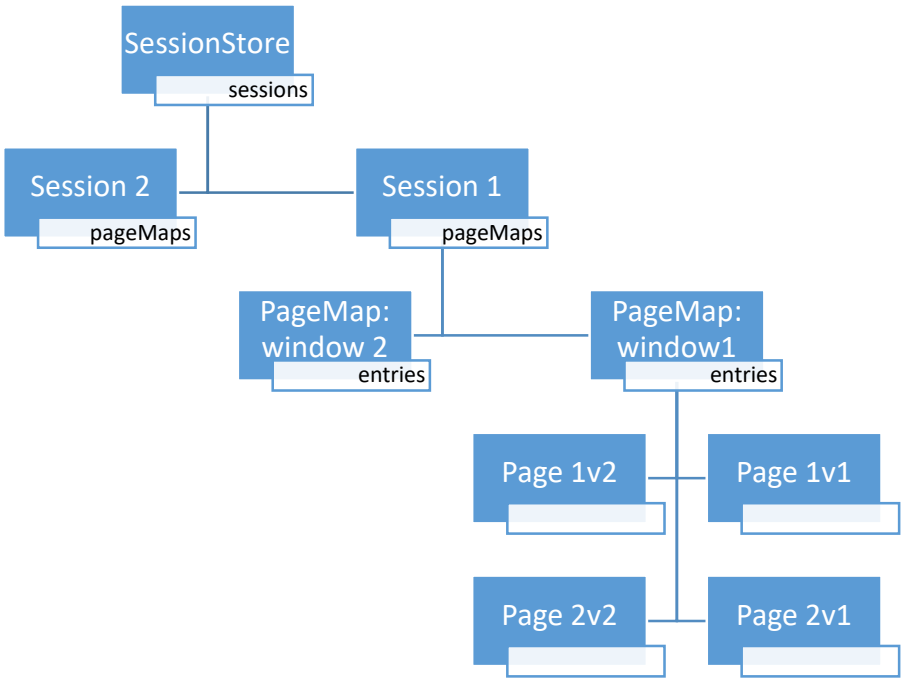
Κάθε διαδικτυακή εφαρμογή διαθέτει ένα και μοναδικό Application στιγμιότυπο. Στην πραγματικότητα το Application αντικείμενο χρησιμοποιείται για να ρυθμίζει πως θα συμπεριφέρεται το Wicket στις διάφορες υπηρεσίες που παρέχει η εφαρμογή. Παρέχει ένα σύνολο από μεθόδους για την διαχείριση των request, sessions κ.τ.λ.

Session

Στο session διατηρούνται τα “ίχνη” του χρήστη, για όση διάρκεια περιηγείται στην εφαρμογή. Για κάθε χρήστη δημιουργείται ένα μοναδικό session στιγμιότυπο. Το session καταστρέφεται είτε όταν ο χρήστης αποσυνδεθεί, είτε όταν εξαντληθεί ο προκαθορισμένος χρόνος αδράνειας (timeout), στην εφαρμογή.

SessionStore

Το session store είναι υπεύθυνο για το που, πότε, και πόσο χρονικό διάστημα θα διατηρηθούν τα δεδομένα του session. Μια συνηθισμένη λειτουργία του store είναι να αποθηκεύει την ενεργή σελίδα στο HttpSession αντικείμενο και τις παλαιότερες να τις διατηρεί σε μια προσωρινή μνήμη (temporary directory) για τη χρήση του κουμπιού επιστροφής. Κάθε εφαρμογή διαθέτει ένα store. Επιπλέον το session store είναι επιφορτισμένο στο να κρατάει τα ίχνη του χρήστη κατά τη διάρκεια περιήγησής του στην εφαρμογή. Στο διάγραμμα 2.3 βλέπουμε ότι το ιστορικό των αιτημάτων αποθηκεύεται σε ένα mapping των σελίδων που έχει επισκεφθεί ο χρήστης. Τα στιγμιότυπα των Page copmonents ομαδοποιούνται σε map λίστες. Το ιστορικό των σελίδων που έχει επισκεφθεί ο χρήστης αποθηκεύεται σε ένα page map ανά session.



Διάγραμμα 2.3: Εννοιολογική σχέση μεταξύ session store, sessions, page maps και pages.

Υπάρχει η περίπτωση να χρειαστούν πολλαπλά page map ανά χρήστη για να υποστηρίξουν την χρήση διαφορετικών browser από τον ίδιο χρήστη.

Request

Σε ένα Request αντικείμενο ενθυλακώνονται πληροφορίες όπως το URL και οι παράμετροί του. Για κάθε request δημιουργείται ένα μοναδικό στιγμιότυπο.

Response

Στο Response αντικείμενο ενθυλακώνονται όλες οι απαραίτητες πληροφορίες που χρειάζονται για να απαντήσουν στο request. Για κάθε response δημιουργείται ένα μοναδικό στιγμιότυπο.

RequestCycle

Το request cycle είναι επιφορτισμένο με την επεξεργασία ενός request, χρησιμοποιώντας το Request και το Response στιγμιότυπο. Από τις κύριες αρμοδιότητες του είναι η ανάθεση του κατάλληλου βήματος επεξεργασίας στον RequestCycleProcessor και η διατήρηση ενός reference του RequestTarget που πρόκειται να εκτελεστεί.

RequestCycleProcessor

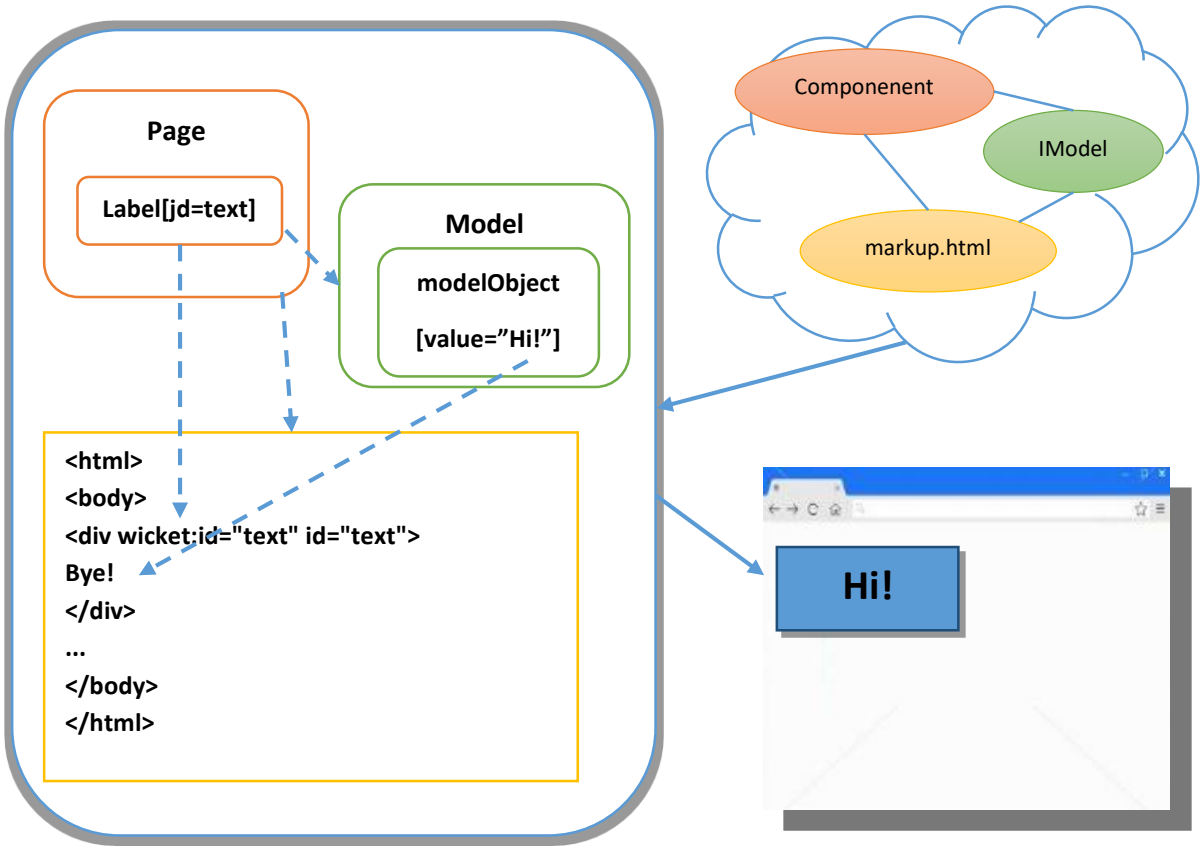
Η RequestCycleProcessor κλάση είναι αρμόδια για να υλοποιεί τα βήματα που απαιτούνται για την επεξεργασία ενός request. Πιο συγκεκριμένα, είναι επιφορτισμένη με το πώς προσδιορίζεται το request target, πως τα γεγονότα (events) διέρχονται μέσα από τον κατάλληλο event handler, και με ποιο τρόπο το αίτημα μεταβιβάζεται.

RequestTarget

Η request target ενθυλακώνει το είδος της επεξεργασίας το οποίο θα εκτελεστεί από το Wicket. Για παράδειγμα ένα request target μπορεί να είναι η αναζήτηση ενός βιβλίου ή το πάτημα του κουμπιού για την επιστροφή στην προηγουμένη σελίδα. Πολλαπλά request targets μπορούν να δημιουργήσουν ένα request, αλλά στο τέλος μόνο ένα χρησιμοποιείται για να διαχειριστεί την απόκριση (response) στον χρήστη.

2.3 MVC αρχιτεκτονική στο Wicket

Τα markup HTML αρχεία αποτελούν τα στατικά κομμάτια της σελίδας. Τα Java components γεμίζουν τα δυναμικά κομμάτια του markup και τα μοντέλα χρησιμοποιούνται από τα components για να πάρουν τα δεδομένα για τα δυναμικά τους μέρη (διάγραμμα 2.4).



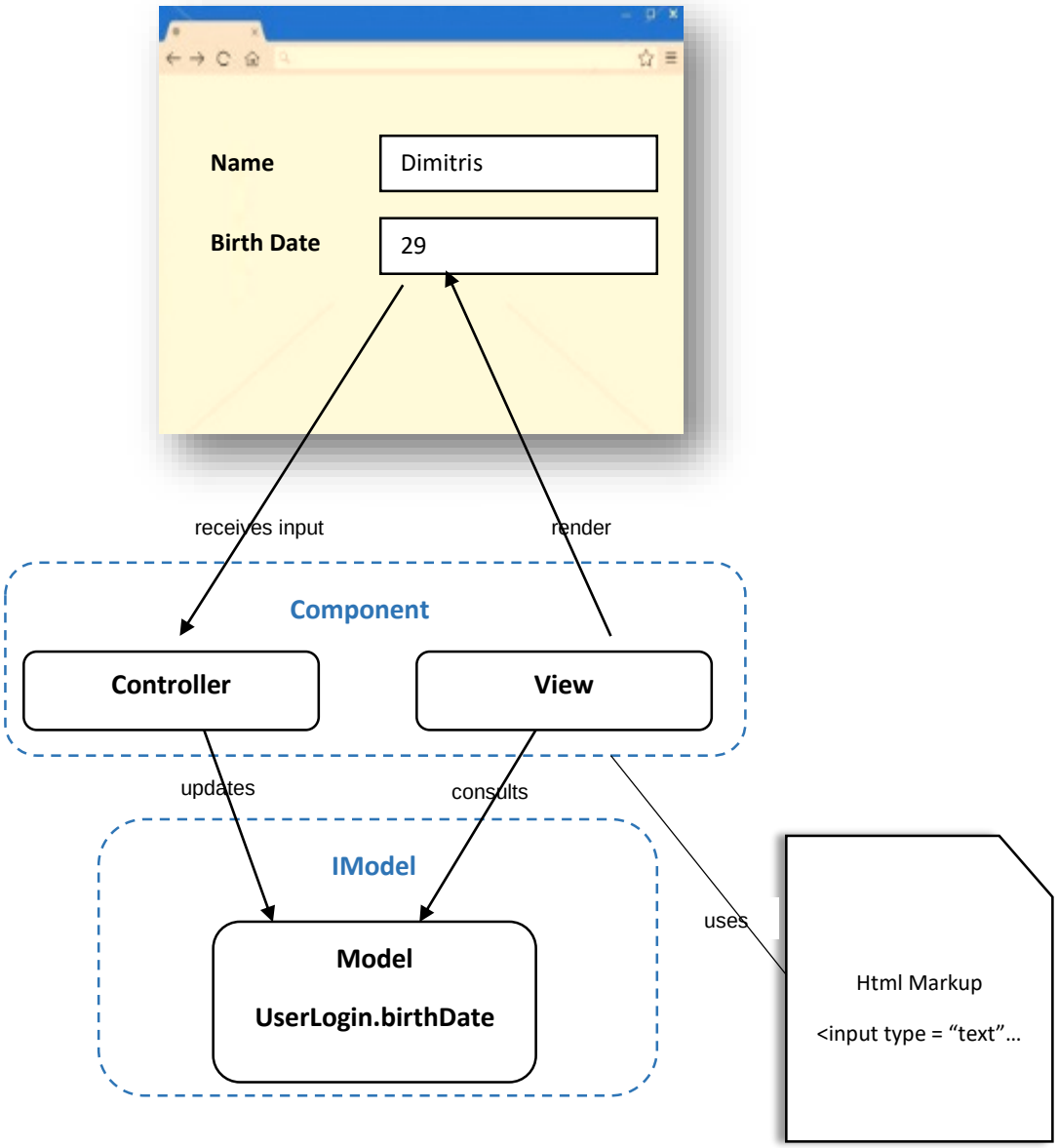
Διάγραμμα 2.4: Σχέση των Components, models, markup στο Wicket.

- Το HTML αρχείο περιέχει αυτό που θα εμφανιστεί στην οθόνη του χρήστη. Τα "wicket:id" tags είναι συνδεδεμένα με τα αντίστοιχα Java components. Μέσα στα " " tags υπάρχει το wicket:id "text".
- Το Label component με αναγνωριστικό "text" συνδέεται με το αντίστοιχο "wicket:id" tag στο HTML.
- Το Label component, χρησιμοποιεί ένα μοντέλο το οποίο παράγει το string "Hi!". Το Label αντικαθιστά το περιεχόμενο του HTML tag (Bye) με την τιμή του μοντέλου με αποτέλεσμα να εμφανίζεται στην οθόνη του χρήστη η γραμματοσειρά "Hi!".

Στο Apache Wicket τα HTML αρχεία δεν περιέχουν λογική, που σημαίνει ότι δεν θα δούμε ποτέ μέσα σε αυτά επαναληπτικές συνθήκες και διαδικασίες. Η λογική

όπως το πότε θα εμφανιστεί ή εξαφανιστεί ένα κουμπί, υλοποιείται από τα components.

Κάθε στοιχείο της MVC τριάδας εκτελεί μια συγκεκριμένη λειτουργία (διάγραμμα 2.5).

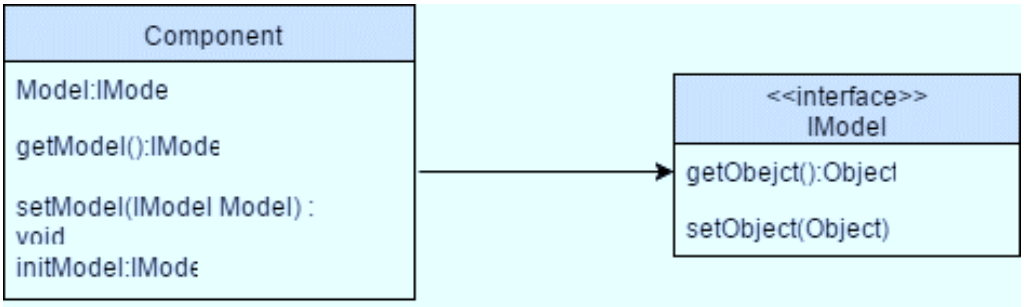


Διάγραμμα 2.5: MVC αρχιτεκτονική στο Wicket.

- **Model:** Το Model αναπαριστά το domain object και την αλληλεπίδραση του με αυτό.
- **View:** Το View είναι υπεύθυνο για το πώς θα εμφανίζεται στην οθόνη του χρήστη το κάθε component. Επιπλέον "ρωτάει" το Model για τυχόν αλλαγές στο μοντέλο του.

- **Controller:** Ο Controller λαμβάνει την είσοδο του χρήστη. Η είσοδος του χρήστη μπορεί να είναι η τιμή ενός textfield, ενός checkbox ή το πάτημα ενός κουμπιού. Το component χρησιμοποιεί αυτά τα δεδομένα για να ενημερώσει το Model.

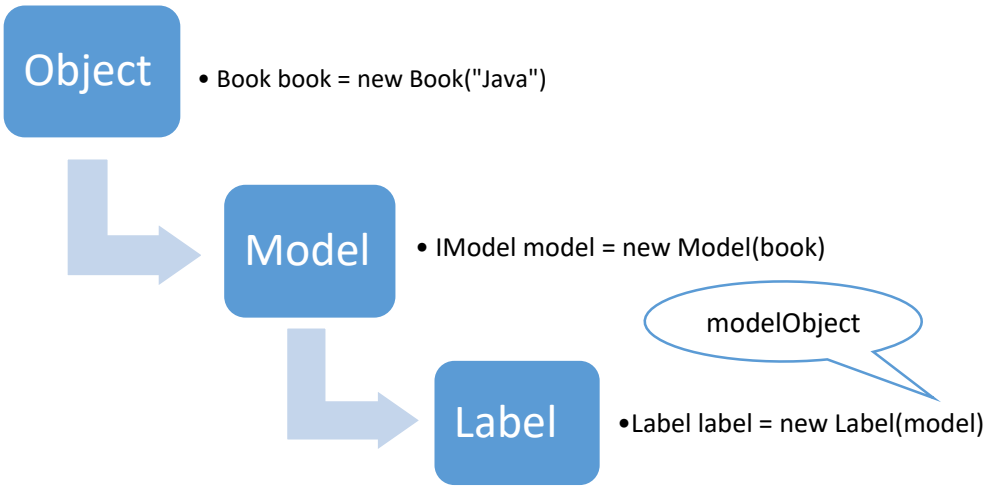
Στις desktop εφαρμογές ο Controller είναι υπεύθυνος για να στέλνει μηνύματα στο View, είτε όταν αντιλαμβάνεται αλλαγές στο μοντέλο, είτε όταν λαμβάνει δεδομένα εισόδου από τον χρήστη. Στις Wicket διαδικτυακές εφαρμογές, είναι αρκετό να ενημερώσεις τα δεδομένα του Model, τα οποία χρησιμοποιούνται από το View, έτσι ώστε όταν το component φορτωθεί στην σελίδα (render) να χρησιμοποιήσει τα νέα δεδομένα.



Διάγραμμα 2.6: Διεπαφή του IModel.

Η δομή της IModel διεπαφής είναι αρκετά απλή. Αποτελείται από δύο μεθόδους την getObject() και την setObject(Object) (διάγραμμα 2.6). Τα components κρατούν μια αναφορά του μοντέλου. Είναι πιθανό ένα component να είναι συνδεδεμένο με περισσότερα από ένα μοντέλα, αλλά συνηθέστερο είναι να συνδέονται με ένα ή κανένα.

Ο όρος “Model” είναι αρκετά συγκεχυμένος καθώς οι περισσότεροι θεωρούν ότι το μοντέλο αφορά τα δεδομένα του εκάστοτε component. Σύμφωνα με το Wicket, το Model θα μπορούσε να χαρακτηριστεί ως εντοπιστής (locators). Το διάγραμμα 2.7, μας δείχνει ότι το Label component έχει σαν όρισμα στον κατασκευαστή του ένα Model. Στην πραγματικότητα έχει το μοντέλο, το οποίο είναι συνδεδεμένο με το αντίστοιχο Object (ModelObject). Όταν το Label φορτωθεί στη σελίδα, καλεί τη getModelObject. Η getModelObject με τη σειρά της καλεί την getObject.



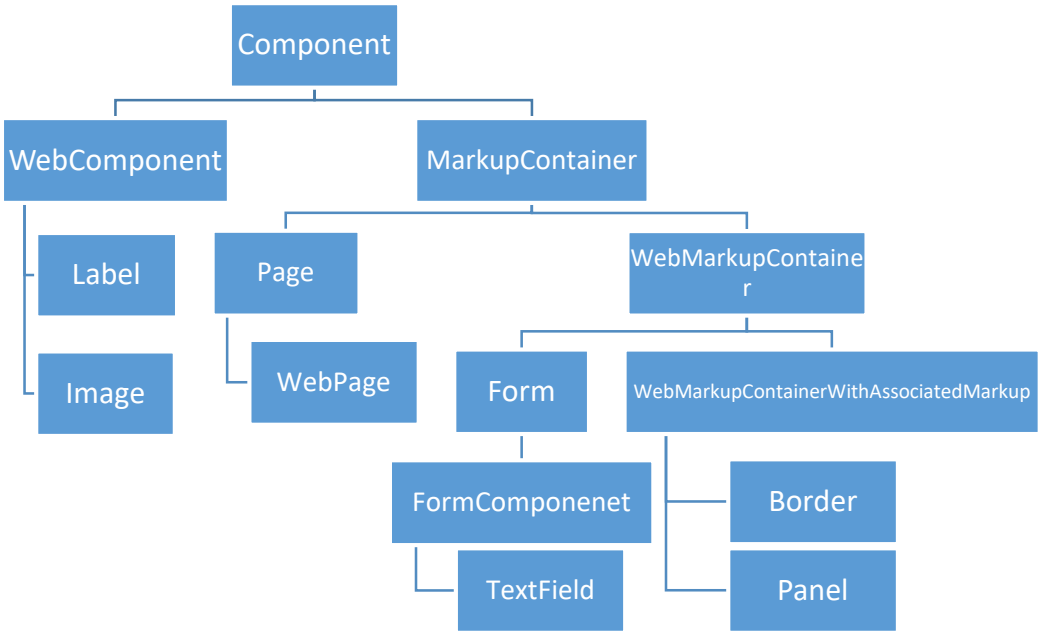
Διάγραμμα 2.7: Ανάλυση model object.

Κεφάλαιο 3

Χαρακτηριστικά του Apache Wicket

3.1 Component

Με τον όρο Component χαρακτηρίζεται οποιοδήποτε αντικείμενο έχει γραφική απεικόνιση και ο χρήστης έχει τη δυνατότητα να αλληλεπιδρά με αυτό. Στο Wicket το component είναι η γονική κλάση από την οποία οι παράγωγες κλάσεις κληρονομούν τα χαρακτηριστικά και τις λειτουργίες της. Στο διάγραμμα 3.1 απεικονίζεται ενδεικτικά η ιεραρχία των component.



Διάγραμμα 3.1: Ιεραρχία των Component σε UML απεικόνιση [https://cwiki.apache.org/confluence/hierarchy].

MarkupContainer

Ο MarkupContainer αποτελείται από έναν πίνακα στον οποίο μπορούν να προστεθούν παιδιά components με τη κλήση της μεθόδου add() ή να αντικατασταθούν με τη replace(). Ένας MarkupContainer “a”, ο οποίος έχει μια υποκλάση “b” και ο “b” ο οποίος έχει τη “c”, καλώντας την a.get(“b.c”) θα μας επιστρέψει το component με id “c”. Ο αριθμός των παιδιών σε έναν MarkupContainer μπορεί να βρεθεί καλώντας την μέθοδο size().

WebMarkupContainer

Είναι ένας container που αποτελείται από component και HTML γλώσσα σήμανσης (markup). Η διαφορά με τον MarkupContainer είναι ότι τα component

υποκλάσεις δηλώνονται μέσα στα ήδη υπάρχοντα component και χρησιμοποιούν το HTML αρχείο του πατέρα τους.

Panel

Το Panel είναι ένα επαναχρησιμοποιήσιμο (reusable) component στο οποίο μπορούν να προστεθούν άλλα components. Είναι υποκλάση του WebMarkupContainerWithAssociatedMarkup και η διαφορά του με τα component του WebMarkupContainer είναι ότι διαθέτει το δικό του HTML markup.

Page

Το Page είναι μια “abstract” κλάση. Ως υποκλάση του MarkupContainer, σε ένα Page είναι δυνατό να περιέχονται διάφορα component ιεραρχικά δομημένα. Η WebPage είναι η μόνη κλάση που κληρονομεί τα χαρακτηριστικά της Page και διαθέτει το δικό της HTML αρχείο.

WebComponent

Είναι μια βασική (base) κλάση για απλά HTML components στην οποία δεν μπορούν να προστεθούν άλλα components.

Form

Το Form component αποτελείται από ένα HTML και ένα Java κομμάτι κώδικα. Η φόρμα υποδέχεται τα στοιχεία που εισάγει ο χρήστης και με το πάτημα του κουμπιού επιβεβαίωσης προωθεί τα στοιχεία για επεξεργασία. Σε μια φόρμα μπορούν να προστεθούν components όπως τα TextField, RadioChoice, CheckBox και Button.

3.1.1 Χαρακτηριστικά των Components

Στη συνέχεια παρουσιάζονται μερικά από τα χαρακτηριστικά των components:

- **Identity** (ταυτότητα): Όλα τα components χαρακτηρίζονται από ένα αναγνωριστικό “id” το οποίο μπορούμε να το ανακαλύψουμε καλώντας την getId() μέθοδο. Το id ενός component είναι μοναδικό μέσα στο MarkupContainer στο οποίο ανήκει αλλά όχι σε ολόκληρη την ιεραρχία του component. Είναι αδύνατον να υπάρχει το ίδιο αντικείμενο σαν υποκλάση για δυο διαφορετικά MarkupContainers.
- **Hierarchy** (ιεραρχία): Όλα τα components έχουν μία πατέρα (parent) κλάση την οποία μπορούν να καλέσουν με την μέθοδο getParents(). Αν ένα component είναι στιγμιότυπο του MarkupContainer ίσως αυτό να έχει παιδιά κλάσεις (children). Η μέθοδος isAncestorOf(Component) επιστρέφει “true”

εάν το παρόν component είναι πρόγονος του component που δίνεται στη παράμετρο.

- **Model** (μοντέλο): Το μοντέλο είναι ένα αντικείμενο το οποίο υλοποιεί την IModel διεπαφή. Η κύρια αρμοδιότητα του component είναι, χρησιμοποιώντας το μοντέλο με το οποίο είναι συνδεδεμένο, να μεταδίδει τις αποκρίσεις (responses) στο HTML. Επιπλέον τα component γνωρίζουν πώς να ενημερώνουν τα μοντέλα τους βασιζόμενα στις πληροφορίες των αιτήσεων (requests). Το component μπορεί να θέσει αλλά και να λάβει μια τιμή από το μοντέλο του μέσω των setModel(IModel Model) και getModel() μεθόδων αντίστοιχα.
- **Versioning** (έκδοση): Όλα τα component μπορούν να υποστηρίξουν versioning. Με το versioning αποθηκεύεται κάθε φορά η κατάσταση του component, έτσι ώστε να μπορεί εύκολα να επανέλθει σε κάποια παλιότερη. Με την isVersioned() μέθοδο, ελέγχουμε αν το component έχει ενεργοποιημένο versioning. Όλα τα component είναι προκαθορισμένα να έχουν ενεργοποιημένο το versioning (δηλαδή true). Στη περίπτωση που θελήσουμε να αλλάξουμε αυτή τη τιμή χρησιμοποιούμε τη μέθοδο setVersioned(boolean).
- **Visibility** (ορατότητα): Τα component μπορεί να είναι μη ορατά στην διαδικτυακή εφαρμογή, Τα μη ορατά components και οι υποκλάσεις τους δεν λαμβάνουν αιτήσεις. Η ορατότητά τους καθορίζεται με τη setVisible(boolean visible) και ελέγχεται με την isVisible().
- **AttributeModifiers** (ιδιότητες): Αντί να αλλαχθεί ο HTML κώδικας ώστε να προστεθεί κάποιο χαρακτηριστικό στο component, αυτό μπορεί να γίνει με Java κώδικα μέσω των attributeModifiers μεθόδων.
- **LifeCycle** (κύκλος ζωής): Ο κύκλος ζωής των components αποτελείται από τα εξής στάδια:
    - **Construction** (δημιουργία): Ένα component όπως και κάθε αντικείμενο στη Java γλώσσα προγραμματισμού δημιουργείται με τη λέξη “new”**.**
    - **Request Handling** (διαχείριση αιτήσεων): Κάθε εισερχόμενο αίτημα διαχειρίζεται και προωθείται από το WicketServlet. Αφού το αίτημα

φτιάσει στο components, ενημερώνει το μοντέλο του και στέλνει την απόκριση.

- **Listener Invocation** (ακροατής): Ο Listener είναι μια μέθοδος που ενεργοποιείται όταν ο χρήστης κάνει κλικ ή επιβεβαιώσει τα στοιχεία μια φόρμας που είναι συνδεδεμένα με κάποιο component.
- **onBeginRequest** (εκκίνηση αιτήματος)**:** Η μέθοδος onBeginRequest() καλείται κατά τη δημιουργία ενός αιτήματος.
- **Form Submit** (υποβολή φόρμας): Αν σε ένα FormComponent γίνει υποβολή των στοιχείων της φόρμας, η επικύρωσή τους γίνετε με το κάλεσμα της μεθόδου FormComponent.validate().
- **Form Model Update** (ενημέρωση μοντέλου φόρμας): Αν η επικύρωση των στοιχείων της φόρμας είναι επιτυχής, τότε το μοντέλο της ενημερώνεται καλώντας τη μέθοδο FormComponent.validate().
- **Rendering** (προώθηση): Κάθε απόκριση που προκύπτει από ένα components γίνεται μέσω της render() μεθόδου.
- **onEndRequest** (τερματισμός αιτήματος): Η μέθοδος onEndRequest() καλείται με την ολοκλήρωση ενός αιτήματος

3.1.2 Δημιουργία component

Στην εικόνα 3.1 παρουσιάζεται η δημιουργία ενός Label component:

```
<html>
    <body>
        <span wicket:id="about-text">Text goes here</span>.
    </body>
</html>
```

Εικόνα 3.1: HTML κώδικας για δημιουργία Label.

Παρατηρούμε 2 βασικά στοιχεία (elements):

- Το Wicket αναγνωριστικό (identifier) wicket:id="about-text"**.**
- Το κείμενο “Text goes here”**.**

Το Wicket αναγνωριστικό πρέπει να έχει ακριβώς το ίδιο όνομα με το component που θα οριστεί στο αντίστοιχο Java αρχείο. Το κείμενο μεταξύ των “span” tags, δηλαδή αυτό που θα εμφανιστεί στο χρήστη, καθορίζεται μόνο από τον Java κώδικα. Το υπάρχον κείμενο αφαιρείται μόλις δημιουργηθεί το Java component (εικόνα 3.2).

```
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.panel.Panel;

public class AboutPanel extends Panel {
    ...

    public AboutPanel(String id) {
        super(id);
        add(new Label("about-text", "The target of..."

    ...
    }
}
```

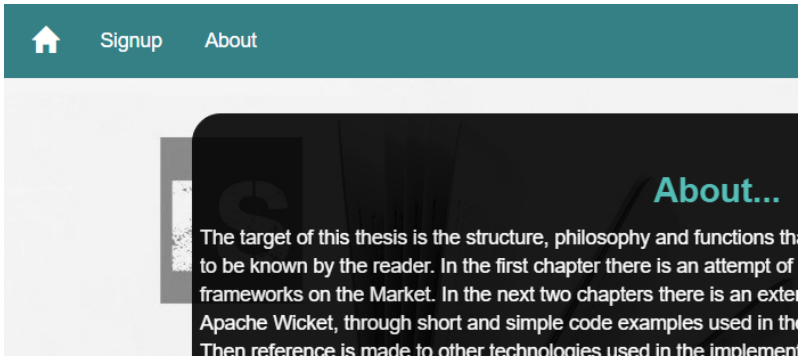
Εικόνα 3.2: Java κώδικας για δημιουργία Label.

Όπως παρατηρούμε (αναφέρθηκε σε προηγουμένη ενότητα) τα δύο αρχεία, Java και HTML πρέπει να έχουν ακριβώς την ίδια ονομασία.

Το Label κατά τη δημιουργία του έχει δυο παραμέτρους:

- “about-text”
- “The target of…”

Η πρώτη παράμετρος είναι το component αναγνωριστικό το οποίο το Wicket χρησιμοποιεί για να αναγνωρίσει το αντίστοιχο Label component στον HTML κώδικα. Η δεύτερη παράμετρος είναι το κείμενο που θα εμφανιστεί στη θέση που έχει οριστεί το Label component (εικόνα 3.3).



Εικόνα 3.3: Απεικόνιση Label στην elibrary εφαρμογή.

3.1.3 Κληρονομικότητα - Επαναχρησιμοποίηση

Μια ιεραρχία κλάσεων (στην περίπτωση μας κάθε κλάση θεωρείται ένα component) έχει τη μορφή δέντρου και διαβάζεται από πάνω προς τα κάτω. Στην κορυφή βρίσκεται η βασική κλάση ενώ όσο προχωράμε προς τα κάτω οι κλάσεις εξειδικεύονται προσθέτοντας χαρακτηριστικά και συμπεριφορές. Κάθε κλάση κληρονομεί χαρακτηριστικά από αυτές που βρίσκονται πάνω από αυτήν ενώ παράλληλα κληροδοτεί σε αυτές που βρίσκονται από κάτω της.

Η επαναχρησιμοποίηση είναι η ικανότητα των προϊόντων λογισμικού να επαναχρησιμοποιούνται, είτε ολικά είτε τμηματικά, στην κατασκευή εφαρμογών. Οι κλάσεις είναι ο κυριότερος μηχανισμός για την εφαρμογή της επαναχρησιμοποίησης σε μια εφαρμογή.

Οι όροι κληρονομικότητα (inheritance) και επαναχρησιμοποίηση (reusability) είναι αλληλένδετοι. Με την αρχή της κληρονομικότητας, δηλαδή τη δημιουργία μιας νέα κλάσης η οποία ονομάζεται παράγωγη (derived) από μία υπάρχουσα (base), επιτυγχάνεται η επαναχρησιμοποίηση κώδικα.

Διάγραμμα 3.2: Markup διάταξη αρχικής σελίδας της elibrary εφαρμογής.

Όπως προαναφέραμε στη Java, μια κλάση μπορεί να κληρονομήσει μια άλλη κλάση “πατέρα”. Το ίδιο σενάριο μπορεί να επιτευχθεί και με τα markup HTML αρχεία. Τα HTML αρχεία των panels components κληρονομούν την HTML διάταξη των container, που κάνουν “extend”.

Η ElibraryPage είναι η αρχική σελίδα της εφαρμογής και περιέχει δυο Panel, το NavigationBarPanel και το FooterPanel. Η LoginPage είναι υποκλάση της ElibraryPage που σημαίνει ότι κληρονομεί οτιδήποτε υπάρχει στην κεντρική σελίδα. Επιπλέον στη LoginPage εισάγεται το LoginPanel (διάγραμμα 3.2).

```
<html>
    <head></head>
    <body>
        <div wicket:id="nav-bar-panel"></div>
        <wicket:child />
        <div wicket:id="footer-panel"></div>
    </body>
</html>
```

Εικόνα 3.4: HTML κώδικας της ElibraryPage.

Παρατηρούμε τρία βασικά στοιχεία (elements):

- wicket:id="nav-bar-panel"
- <wicket:child />
- wicket:id="footer-panel"

Η HTML markup κληρονομικότητα επιτυγχάνεται με τη χρήση δυο συγκεκριμένων tags (εικόνα 3.4). Το <wicket:child /> χρησιμοποιείται από τον “πατέρα” markup container για να ορίσει το σημείο στο οποίο θα τοποθετηθεί ο HTML κώδικας του “παιδιού”. Αντίστοιχα στο "παιδί” (στη συγκεκριμένη περίπτωση στην LoginPage) με τα tags <wicket:extend> και </wicket:extend> δηλώνεται που ξεκινά και τελειώνει ο HTML κώδικας (εικόνα 3.5).

```
<html>
    <body>
        <wicket:extend>
            <div wicket:id="login"></div>
        </wicket:extend>
    </body>
</html>
```

Εικόνα 3.5: HTML κώδικας της LoginPage.

Παρατηρούμε τρία βασικά στοιχεία (elements):

- <wicket:extend>
- wicket:id="login"
- </wicket:extend>

```
<html>
    <head></head>
    <body>
        <div wicket:id="nav-bar-panel"></div>
        <wicket:child>
            <wicket:extend>
                <div wicket:id="login"></div>
            </wicket:extend>
        </wicket:child>
        <div wicket:id="footer-panel"></div>
    </body>
</html>
```

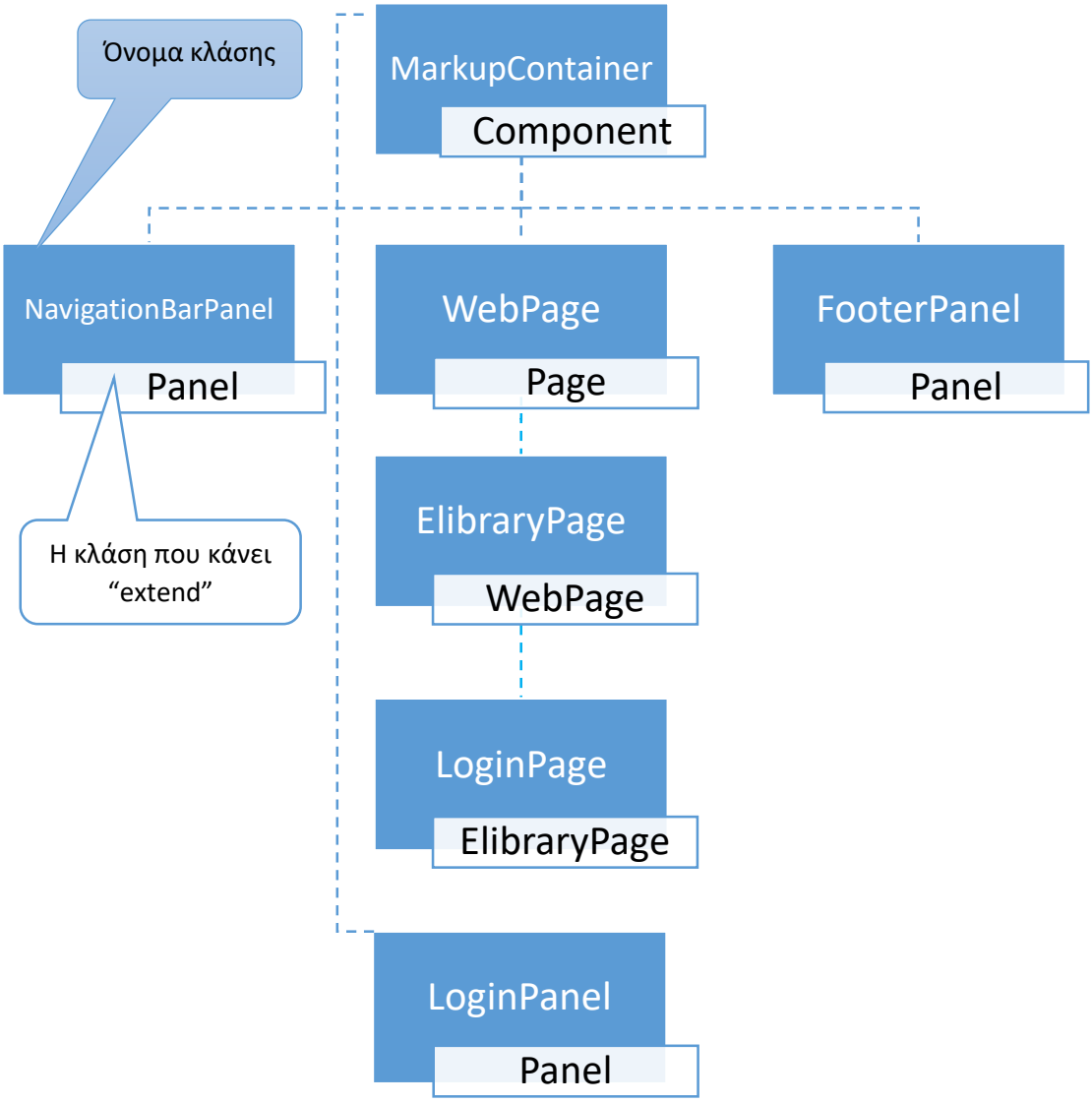
Εικόνα 3.6: Τελικός HTML κώδικας αρχικής σελίδας ElibraryPage.

Όπως παρατηρούμε (εικόνα 3.6) ανάμεσα στα <wicket:child> tags έχει τοποθετηθεί το περιεχόμενο των <wicket:extend> tags. Γυρίζοντας πάλι στο markup της αρχικής σελίδας βλέπουμε δυο tags το wicket:id="nav-bar-panel" και το wicket:id="footer-panel". Με αυτό τον τρόπο, δηλώνεται ότι στην αρχή και στο τέλος της σελίδας εισάγονται δυο Panel components. Όπως αναφέρθηκε σε προηγούμενη παράγραφο τα Panel διαθέτουν τον δικό τους HTML κώδικα. Ένα Panel μπορεί να εισαχθεί σε οποιαδήποτε σημείο ενός Page component. Στην κεντρική σελίδα, της εφαρμογής που υλοποιήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας, εισάγεται το NavigationBarPanel και το FooterPanel. Επιπλέον στην υποκλάση της ElibraryPage, δηλαδή την LoginPage, εισάγεται το LoginPanel.

```
<html>
<wicket:panel>

    ...

</wicket:panel>
</html>
```

Εικόνα 3.7: HTML κώδικας της LoginPanel.

Οτιδήποτε δηλώνεται μέσα στα <wicket:panel> και </wicket:panel> tags (εικόνα 3.7), θα εμφανιστεί στο πάτερα markup στο σημείο στο οποίο έχει καθοριστεί το Panel. Ότι δηλώνεται εκτός των παραπάνω tags θα αγνοηθεί από την Wicket εφαρμογή και δεν θα εμφανιστεί ποτέ.



Διάγραμμα 3.3: Ιεραρχική απεικόνιση των component κλάσεων.

Παρατηρούμε (διάγραμμα 3.3) ότι η ElibraryPage είναι υποκλάση της WebPage, η οποία εν συνεχεία κάνει "extend" την abstract κλάση Page. Στον κατασκευαστή της εισάγονται δύο Panel components (εικόνα 3.8). Το NavigationBarPanel είναι το όνομα της κλάσης του component και το "nav-bar-panel" το wicket id έτσι όπως είναι δηλωμένο στον HTML κώδικα της ElibraryPage. Ομοίως, ως FooterPanel δηλώνεται το όνομα τη κλάσης και ως "footer-panel" το αντίστοιχο wicket id στο HTML αρχείο της κεντρικής σελίδας.

```
import org.apache.wicket.markup.html.WebPage;
import dskarpetis.elibrary.ui.authentication.LoginPage;
import dskarpetis.elibrary.ui.footer.FooterPanel;
import dskarpetis.elibrary.ui.navigation.NavigationBarPanel;

public class ElibraryPage extends WebPage {
    ...

    public ElibraryPage() {
        // add navigation panel in base page
        add(new NavigationBarPanel("nav-bar-panel"));
        // add footer panel in base page
        add(new FooterPanel("footer-panel"));
    }
    ...
}
```

Εικόνα 3.8: Java κώδικας της ElibraryPage.

Η LoginPage κλάση, κληρονομεί τις ιδιότητες και τα χαρακτηριστικά της ElibraryPage.

```
import org.apache.wicket.Page;

public class LoginPage extends ElibraryPage {
    ...

    public LoginPage() {
        add(new LoginPanel("login"));
    }
    ...
}
```

Εικόνα 3.9: Java κώδικας LoginPage.

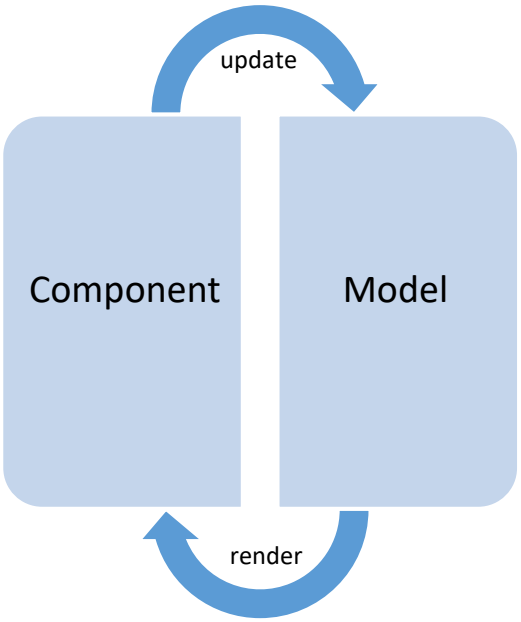
Στον κατασκευαστή της LoginPage (εικόνα 3.9) εισάγεται το LoginPanel με το “login” να είναι το wicket id στον HTML κώδικά της. Στην εικόνα 3.10, παρατηρούμε πως το NavigationBarPanel κάνει “extend” τη κλάση Panel.

```
import org.apache.wicket.markup.html.panel.Panel;
import dskarpetis.elibrary.ui.authentication.LoginPage;

public class NavigationBarPanel extends Panel {
    ...

    public NavigationBarPanel(final String id) {
    ...

    }
}
```

Εικόνα 3.10: Java κώδικας του NavigationBarPanel.

3.2 Model

Σε προηγούμενο κεφάλαιο μιλήσαμε για τη Model View Controller αρχιτεκτονική σχεδίασης. Τα Wicket components αποτελούν το View και το Controller κομμάτι, ενώ τα domain αντικείμενα της εφαρμογής αντιπροσωπεύουν το ρόλο του Model. Τα Wicket model από τη μία, διαχωρίζουν το domain (UserLogin, Book, Author, Publisher) από το view (TextField, Label, Panel) επίπεδο, και από την άλλη τα δένουν μεταξύ τους δημιουργώντας μια συνεκτική και πολυεπίπεδη δομή. Τα model επιτρέπουν στα component, όταν αυτά φορτώνονται στη σελίδα, να “τραβούν” τα δεδομένα τους. Επιπλέον όταν συμβεί οποιαδήποτε αλλαγή στο component τα νέα δεδομένα αποθηκεύονται στο model (διάγραμμα 3.4).



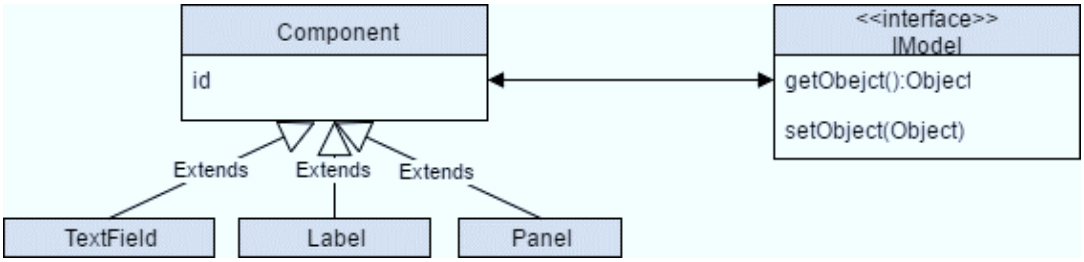
Διάγραμμα 3.4: Σχέση μεταξύ component και model.

Τα model είναι από τα σημαντικότερα κομμάτια του Wicket διαδικτυακού εργαλείου, καθώς επηρεάζουν άμεσα την απόδοση και την ταχύτητα της εφαρμογής. Η χρήση των model βοηθά τον προγραμματιστή να γράφει λιγότερες γραμμές κώδικα, που έχει ως συνέπεια την βελτίωση της επεκτασιμότητας και συντηρησιμότητας της εφαρμογής. Επειδή το model βρίσκεται ανάμεσα στο view και το domain επίπεδο, τα δεδομένα μπορούν να επεξεργαστούν είτε πριν αυτά εμφανιστούν στην οθόνη, είτε πριν το domain αντικείμενο δεχθεί κάποια είσοδο του χρήστη. Όλα τα components έχουν σαν μέλος ένα model.

Η IModel διεπαφή αποτελείται από δύο μεθόδους:

- **getObject()**: επιστρέφει τη τιμή του μοντέλου.
- **setObject()**: θέτει τη τιμή του μοντέλου.

Ανάλογα με την υλοποίηση του model, η τιμή του μπορεί να δηλωθεί τοπικά και να είναι αυτόνομη εντός του μοντέλου ή μπορεί να οριστεί δυναμικά μέσω του domain αντικειμένου. Η υλοποίηση του model είναι αρκετά ευέλικτη και σου επιτρέπει να παίρνεις τα δεδομένα του domain ανεξαρτήτως αν αυτό βρίσκεται στη βάση, στο session, στο web service ή οπουδήποτε αλλού. Το component δεν το ενδιαφέρει από πού θα πάρει τα δεδομένα ή που αυτά είναι αποθηκευμένα, το μόνο που χρειάζεται είναι να μπορεί να καλεί τις getObject() και setObject() μεθόδους (διάγραμμα 3.5).



Διάγραμμα 3.5: Απεικόνιση της σχέσης μεταξύ component και IModel.

Στη πραγματικότητα το model αντικατοπτρίζει την τιμή του component. Αντί να ρωτάς το component για την τιμή του, ρωτάς απευθείας το μοντέλο και αντί να θέτεις απευθείας τα δεδομένα στο component, αλλάζεις τη τιμή του μοντέλου. Ο προγραμματιστής του Wicket framework, κρατώντας μια αναφορά του μοντέλου (ή γνωρίζοντας που βρίσκεται το μοντέλο) έχει πρόσβαση στα δεδομένα του component που συνδέονται με αυτό. Το Wicket παρέχει διάφορες υλοποιήσεις του model που ο προγραμματιστής μπορεί να επιλέξει ανάλογα με τι ανάγκες του. Παρακάτω παρουσιάζονται οι σημαντικότερες υλοποιήσεις της IModel διεπαφής (πίνακας 3.1).

Πίνακας 3.1: Οι κυριότερες υλοποιήσεις της IModel διεπαφής.

Model	Περιγραφή
Model	Το πιο απλό μοντέλο, χρησιμοποιείται κυρίως για την αποθήκευση στατικού περιεχομένου.

PropertyModel	Χρησιμοποιεί expression, για να έχει δυναμική πρόσβαση στην ιδιότητα ενός domain αντικειμένου.
CompoundPropertyModel	Χρησιμοποιεί τα components αναγνωριστικά σαν expressions για να συνδέσει τα components με το αντίστοιχο domain αντικείμενο.
LoadableDetachableModel	Abstract μοντέλο για γρήγορη δημιουργία detachable μοντέλων.
StringResourceModel	Μοντέλο το οποίο ανακτά πόρους από ένα property αρχείο.

3.2.1 Simple Model

Κάθε φορά που γράφουμε τον κώδικα (εικόνα 3.11):

```
...
new Label ("name", book.getName());
...
```

Εικόνα 3.11: Δημιουργία Label Component με χρήση απλού model.

Στο παρασκήνιο συμβαίνει (εικόνα 3.12):

```
...
new Label ("name", new Model ( book.getName() ));
...
```

Εικόνα 3.12: Δημιουργία Label Component στο παρασκήνιο.

Η πρώτη παράμετρος του Label component είναι το wicket id, το οποίο συνδέεται με το tag στο αρχείο HTML. Η δεύτερη παράμετρος είναι τα δεδομένα του μοντέλου, τα οποία θα εμφανιστούν στο σημείο που έχει οριστεί στο markup αρχείο.

Ο κατασκευαστής (εικόνα 3.13, κατ.1) παίρνει την String παράμετρο και την ενσωματώνει σε ένα model πριν καλέσει τον super κατασκευαστή (εικόνα 3.13, κατ.2). Με αυτόν τον τρόπο ο προγραμματιστής περνά άμεσα το String στον κατασκευαστή χωρίς να χρειάζεται ο ίδιος να συνδέσει το object με το μοντέλο.

```
public class Label extends WebComponent {
    public Label(String id, String text) {
        this(id, new Model(text));
    }
    public Label(String id, IModel model) {
        super(id, model);
    }
}
```

κατασκευαστής 1

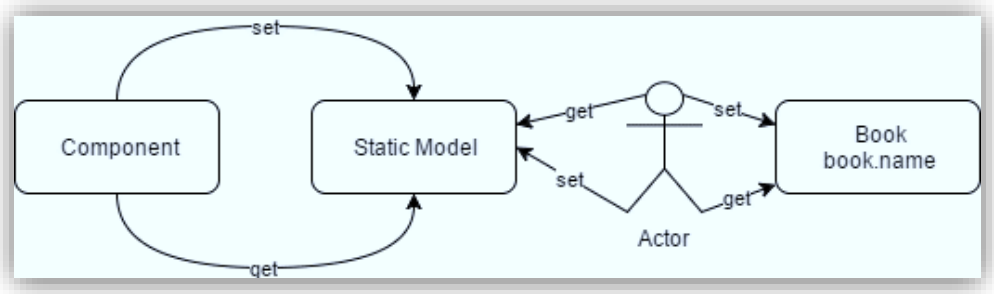
κατασκευαστής 2

Εικόνα 3.13: Κατασκευαστής του Label component στο Wicket.

Η παραπάνω προσέγγιση λειτουργεί μόνο για String παραμέτρους. Αυτό αποτελεί πρόβλημα όταν θέλουμε να εμφανιστεί στο Label component ένας διαφορετικός τύπος από το String (π.χ. Date).

Ένα δεύτερο πολύ σημαντικό πρόβλημα είναι οι null τιμές. Αν το book αντικείμενο είναι null ή η getName() μέθοδος επιστρέψει null τότε στον χρήστη θα επιστραφεί NullPointerException, γεγονός το οποίο δεν το θέλουμε σε καμία περίπτωση.

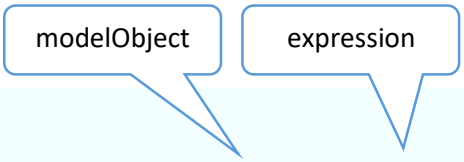
Τρίτο και σημαντικότερο πρόβλημα είναι ότι το String στην Java είναι αμετάβλητο (immutable). Ακόμα και αν η book.getName() αλλάξει τιμή, τα δεδομένα του μοντέλου θα μείνουν αμετάβλητα. Αυτό έχει σαν αποτέλεσμα, αν η σελίδα ανανεωθεί, να εμφανιστούν τα παλιά δεδομένα. Τα μοντέλα των οποίων η τιμή δεν αλλάζει είναι γνωστά ως στατικά μοντέλα (διάγραμμα 3.6).



Διάγραμμα 3.6: Απεικόνιση στατικού μοντέλου.

3.2.2 Property models

Η κλάση PropertyModel, επιτρέπει στο μοντέλο να έχει πρόσβαση σε μια συγκεκριμένη ιδιότητα του αντικειμένου με το οποίο είναι συνδεδεμένο. Η ιδιότητα αυτή είναι προσβάσιμη με τη χρήση μια απλής έκφρασης (expression). Για παράδειγμα η έκφραση “name” δηλώνει την ιδιότητα name και η έκφραση “book.name” σημαίνει η ιδιότητα name του αντικειμένου book (εικόνα 3.14).




```
...
 new TextField<String> ("name", new PropertyModel<String>(book, "name"));
...
```

Εικόνα 3.14: Δημιουργία TextField Component με χρήση PropertyModel.

Όταν το framework "ρωτήσει" το model για τη τιμή του, αυτό θα χρησιμοποιήσει την έκφραση για να αποκτήσει πρόσβαση στην ιδιότητα του αντικείμενου (modelObject). Αν υποθέσουμε ότι η POJO κλάση έχει τη μορφή της εικόνας 3.15, τότε με το expression "name" μπορούμε να έχουμε πρόσβαση στην ιδιότητα name του αντικειμένου Person μέσω της μεθόδου getName().

```
class Book
{
  private String name;

  Book(String name)
  {
    this.name = name;
  }

  String getName()
  {
    return name;
  }
}
```

Εικόνα 3.15: Book domain αντικείμενο.

Ο δεύτερος constructor του PropertyModel είναι διαθέσιμος για μοντέλα τα οποία θέλουν να μετατρέψουν τη τιμή της ιδιότητας του αντικειμένου, στο τύπο της αρεσκείας τους (εικόνα 3.16).

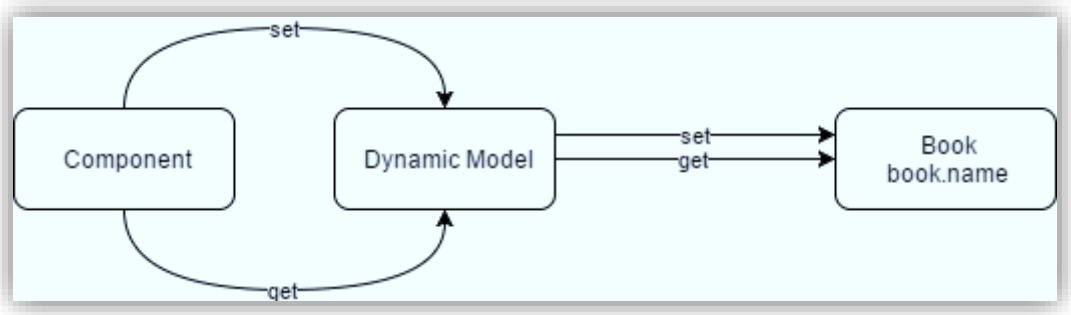
```
public PropertyModel(final Object modelObject, final String expression, Class propertyType)
```

Εικόνα 3.16: Κατασκευαστής του propertyModel με παράμετρο μετατροπής.

Το PropertyModel μετατρέπει αυτομάτως τη τιμή του component στον τύπο της κλάσης που είναι δηλωμένο στην παράμετρο (π.χ. Integer.class).

Ένα σημαντικό πλεονέκτημα του PropertyModel είναι ότι προστατεύει την εφαρμογή από NullPointerException (null safe). Στη περίπτωση που κάποια τιμή είναι null αυτομάτως θα μετατραπεί σε empty.

Με τη χρήση του PropertyModel είναι πολύ πιο εύκολη η ενημέρωση των Java αντικειμένων μέσα στη φόρμα. Δεν είναι αναγκαίο να ρωτήσεις το component για μία τιμή και να την ορίσεις στο domain αντικείμενο, το μόνο που χρειάζεται είναι να συνδέσεις (bind) τη domain κλάση με το αντίστοιχο component της φόρμας χρησιμοποιώντας ένα property μοντέλο. Τα στιγμιότυπα των property model είναι δυναμικά (διάγραμμα 3.7).



Διάγραμμα 3.7: Απεικόνιση δυναμικού μοντέλου.

3.2.3 CompoundPropertyModel

Συνήθως τα component αναγνωριστικά έχουν την ίδια ονομασία με τις ιδιότητες (properties) των domain αντικείμενων. Για παράδειγμα όταν θέλουμε να εμφανίσουμε τα χαρακτηριστικά ενός Book αντικειμένου, δίνουμε ονομασίες στα component αναγνωριστικά όπως name, author, publisher και isbn. Αυτή η προσέγγιση είναι σωστή καθώς δίνει έμφαση στην έννοια του component, στα HTML και Java αρχεία, κάνοντας την εφαρμογή πιο κατανοητή.

```
...

add(new Label("name", new PropertyModel(book, "name")));
add(new Label("author", new PropertyModel(book, "author")));
...
```

Εικόνα 3.17: Δημιουργία components με PropertyModel.

Όπως παρατηρούμε με τα property model, τα component αναγνωριστικά έχουν ακριβώς την ίδια ονομασία με τα properties του book αντικειμένου (εικόνα 3.17).

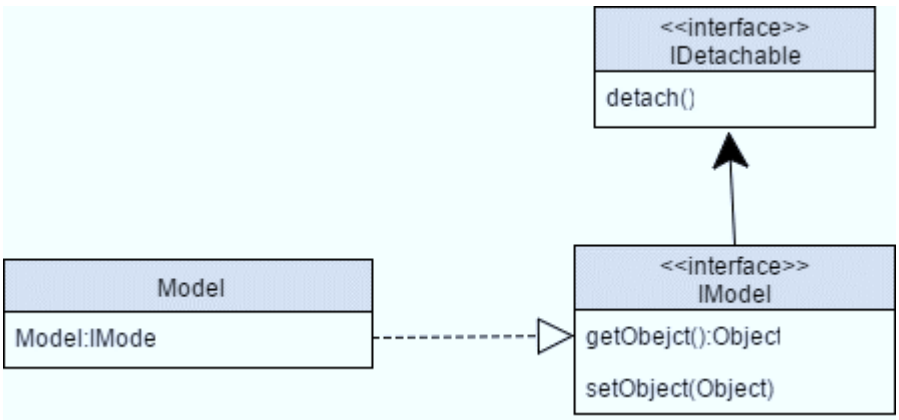
Αρχικά συνδέουμε το μοντέλο του πατέρα component (π.χ. panel, form) σε ένα CompoundPropertyModel το οποίο περικλείει το Book αντικείμενο. Στη συνέχεια αφαιρούμε τα property model από το κάθε Label component. Όταν ένα component χρειαστεί τη τιμή του μοντέλου του, ενώ δεν είναι συνδεδεμένο με κάποιο μοντέλο τότε θα χρησιμοποιήσει το CompoundPropertyModel του πατέρα component. Το component χρησιμοποιεί το wicket αναγνωριστικό σαν property expression με το αντικείμενο που είναι συνδεδεμένο το μοντέλο του πατέρα component (εικόνα 3.18). Ίσως σημαντικότερο πλεονέκτημα του CompoundPropertyModel είναι ότι μειώνει αρκετά και κάνει πιο ευανάγνωστο τον κώδικα της εφαρμογής μας.

```
...
setModel(new CompoundPropertyModel(book));
add(new Label("name"));
add(new Label("author"));
...
```

Εικόνα 3.18: Δημιουργία components με CompoundPropertyModel.

3.2.4 LoadableDetachableModel

Κατά την ολοκλήρωση ενός αιτήματος (request) και μετά τη φόρτωση της σελίδας στον περιηγητή του χρήστη, το Wicket καλεί μια αποδεσμευτική “detach” συνθήκη κατά την οποία όλα τα components και τα μοντέλα τα οποία παίρνουν μέρος στο request έχουν την επιλογή να “καθαρίσουν” οποιαδήποτε πληροφορία. Η detaching προσέγγιση εκτελείται για να ελαχιστοποιήσει το μέγεθος της μνήμης του state της εφαρμογής και να αποδεσμεύσει τις αναφορές (references) των μη serializable αντικειμένων. Όταν το state της εφαρμογής σειριοποιηθεί στον server, δεν θα συμπεριληφθούν σε αυτό αντικείμενα που δεν έχουν υλοποιήσει τη Serialization διεπαφή. (διάγραμμα 3.8).



Διάγραμμα 3.8: Διεπαφή του IDetachabe.

3.2.5 StringResourceModel

Το StringResourceModel έχει σαν παραμέτρους στον κατασκευαστή του, ένα resource key, ένα component και προαιρετικά έναν πίνακα από αντικείμενα (εικόνα 3.19).

```
public StringResourceModel(final String resourceKey, final Component component,
final IModel model)
public StringResourceModel(final String resourceKey, final Component component,
final IModel model, final Object[] parameters)
```

Εικόνα 3.19: Κατασκευαστές του StringResourceModel.

Το resourceKey χρησιμοποιείται για να βρει κάποιο resource σε ένα property αρχείο. Το property αρχείο που θα χρησιμοποιηθεί εξαρτάται από το component. Ένα παράδειγμα κατασκευής ενός Label component με χρήση του StringResourceModel φαίνεται στην εικόνα 3.20.

```
add(new Label("birthdate", new StringResourceModel("label.birthdate", this)));
```

Εικόνα 3.20: Δημιουργία Label με χρήση StringResourceModel.

Όταν το Label γίνει render, στη σελίδα θα εμφανιστεί το κείμενο “Ημερομηνία γέννησης (εικόνα 3.21).

```
label.birthdate= Ημερομηνία γέννησης
```

Εικόνα 3.21: Κείμενο απεικόνισης του Label.

3.3 State management

Το Wicket γεφυρώνει το κενό ανάμεσα στο stateless HTTP πρωτόκολλο και τη stateful δομή της Java. Σε αυτό το σημείο ας κάνουμε μια παρένθεση για να εξηγήσουμε τους όρους stateless και statefull.

Τον όρο Stateless τον χρησιμοποιούμε για να δείξουμε ότι δεν συγκρατείται καμία πληροφορία μεταξύ του διακομιστή (server) και του πελάτη (client) μετά το πέρας της επικοινωνίας τους.

Το HTTP πρωτόκολλο είναι το πιο γνωστό παράδειγμα stateless διασύνδεσης. Ο πελάτης στέλνει αίτημα στην εφαρμογή, ο διακομιστής το επεξεργάζεται και στέλνει απόκριση πίσω στον πελάτη. Με την ολοκλήρωση της διαδικασίας μεταξύ πελάτη και διακομιστή χάνεται και οποιαδήποτε πληροφορία για την μεταξύ τους επικοινωνία. Κάθε επικοινωνία είναι μοναδική και δεν σχετίζεται με τις προηγούμενες και τις επόμενες.

Σε μια statefull εφαρμογή ηλεκτρονικού καταστήματος ο χρήστης μπορεί να περιηγηθεί, να προσθέσει προϊόντα στο καλάθι αγορών, και τέλος να ξανά γυρίσει στην αρχική σελίδα. Το framework φροντίζει ώστε να παραμείνουν τα προϊόντα στο καλάθι αγορών και όταν θελήσει ο πελάτης, να προχωρήσει στη αγορά του. Το παραπάνω παράδειγμα μας δείχνει πως μπορεί να αποθηκευτεί πληροφορία κατά τη διάρκεια συνόδου ενός χρήστη με την εφαρμογή.

Τον όρο Statefull τον χρησιμοποιούμε για να δείξουμε ότι διατηρούνται πληροφορίες μεταξύ του διακομιστή (server) και του πελάτη (client) μετά το πέρας της επικοινωνίας τους.

Η λύση σε αυτό το πρόβλημα είναι η υλοποίηση ενός προγραμματιστικού μοντέλου το οποίο κρύβει την stateless ιδιότητα του HTTP πρωτοκόλλου (state management). Το μόνο που πρέπει να κάνει ο προγραμματιστής είναι να αφοσιωθεί στη λύση των λογικών προβλημάτων, που μπορεί να αντιμετωπίσει η κατασκευή μιας web εφαρμογής και να αφήσει το framework να διαχειριστεί το state.

3.4 Separations of presentation and logic

Το κυριότερο χαρακτηριστικό του Apache Wicket framework είναι ο διαχωρισμός της εμφάνισης από την λογική (separations of presentation and logic). Για κάθε στοιχείο που εισάγεται στο HTML, υπάρχει το αντίστοιχο component στο Java αρχείο.

- **Μόνο Java:** Το Wicket εκμεταλλεύεται πλήρως τις δυνατότητες και τα πλεονεκτήματα της γλώσσας προγραμματισμού Java. Μπορείς να δημιουργήσεις ένα component με το “new” και να κληρονομήσεις τις λειτουργίες και την συμπεριφορά ενός “πατέρα” component με το “extend”.

- **Μόνο HTML:** Η εμφάνιση της εφαρμογής καθορίζεται στο HTML αρχείο. Ένα HTML αρχείο περιέχει στατικό markup κώδικα και wicket ids, τα οποία αντιστοιχίζονται με τα component στο Java κώδικα.

Με τον διαχωρισμό της λογικής από την παρουσίαση ο κώδικας που παράγεται είναι πιο κατανοητός και καθαρός. Ο προγραμματιστής δεν είναι απαραίτητο να γνωρίζει και Java και HTML. Η κατασκευή της εφαρμογής μπορεί να ανατεθεί σε άτομα που ειδικεύονται είτε στο back end είτε στο front end προγραμματισμό αντίστοιχα για Java και HTML.

3.5 Ασφάλεια

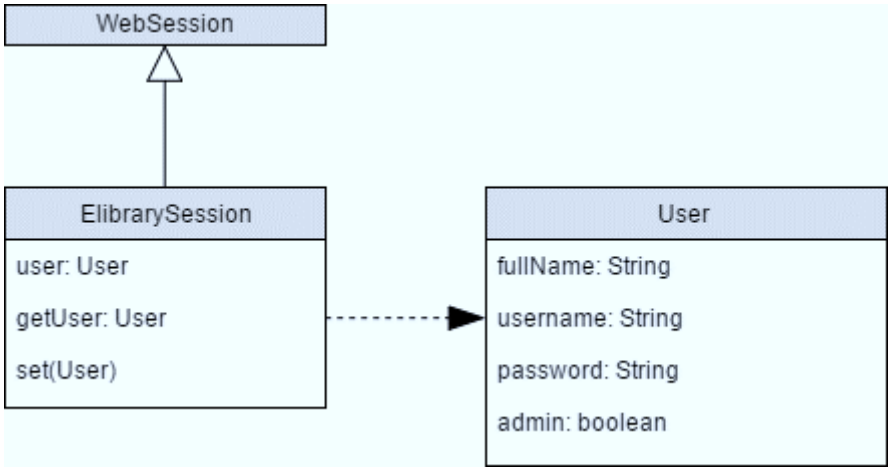
Κατά το σχεδιασμό μιας διαδικτυακής εφαρμογής πρέπει να δίνετε μεγάλο βάρος στους τομείς στους οποίους η εφαρμογή μπορεί να είναι ευάλωτη σε επιθέσεις. Για παράδειγμα, ζητήματα επικύρωσης δεδομένων εισόδου, αυθεντικοποίησης και εξουσιοδότησης. Παρακάτω παρουσιάζονται οι κυριότερες λειτουργίες που παρέχει το Wicket για την ασφαλή θωράκιση των διαδικτυακών εφαρμογών.

3.5.1 Page version

Τα στιγμιότυπα της σελίδας αποθηκεύονται στον server. Όταν ο χρήστης πατήσει ένα link για μια συγκεκριμένη σελίδα, στην πραγματικότητα το Wicket ψάχνει για ένα συγκεκριμένο αριθμό σελίδας, έκδοσης (version) και εν τέλη στο κατάλληλο component το οποίο περιέχεται στο request του χρήστη. Για να παραβιάσει κάποιος την εφαρμογή πρέπει να καταφέρει να “υποκλέψει” το session, να μαντέψει το σωστό αριθμό σελίδας, το σωστό αριθμό έκδοσης και τέλος το path το οποίο θα σχετίζεται με το κατάλληλο component.

3.5.2 Αποθήκευση κινήσεων του χρήστη

Δημιουργούμε μια User κλάση στην οποία κρατάμε τις πληροφορίες των χρηστών. Μέσα στη User αποθηκεύουμε το ονοματεπώνυμο, ένα μοναδικό όνομα χρήστη, τον κωδικό και τέλος ένα πεδίο που δείχνει αν ο χρήστης είναι διαχειριστής. Όταν ο χρήστης κάνει μία επιτυχή σύνδεση με την εφαρμογή, το User αντικείμενο αποθηκεύεται στο Wicket Session. Στην υποκλάση της WebSession, αποθηκεύουμε μια αναφορά (reference) του χρήστη που έχει συνδεθεί με την εφαρμογή (διάγραμμα 3.9).



Διάγραμμα 3.9: Κλάσεις WebSession, ElibrarySession και User.

Αν αυτή η αναφορά ισούται με null, σημαίνει ότι ο χρήστης δεν είναι αυθεντικοποιημένος.

```
public class ElibrarySession extends WebSession {

    private User user;

    public static ElibrarySession get() {
        return (ElibrarySession) Session.get();
    }

    public ElibrarySession (Application application) {
        super (application);
    }

    public boolean isAuthenticated() {
        return (user != null);
    }

    public UserLogin getUser() {
        return user;
    }
}
```

Εικόνα 3.22: Κλάση που υλοποιεί την WebSession.

Το Wicket δημιουργεί αυτόματα sessions για τους χρήστες. Υπερφορτώνοντας (overriding) την newSession (εικόνα 3.23) δημιουργούνται ξεχωριστά sessions για κάθε χρήστη.

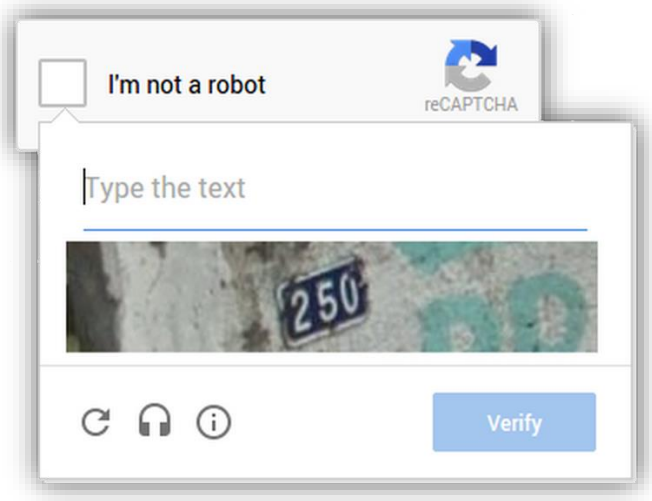
```
@Override
public Session newSession(Request request, Response response) {
    retuen new ElibrarySession(request);
}
```

Εικόνα 3.23: Μέθοδος newSession για την αποθήκευση του χρήστη.

3.5.3 Χρήση Captcha τεχνολογίας

Το Captcha (Completely Automated Public Turing test to tell Computers and Humans Apart) [http://www.captcha.net/] θα μπορούσε να παρομοιαστεί με τεστ Turing, που έχει ως στόχο να επιβεβαιώσει ότι η απάντηση σε αυτό προέρχεται από άνθρωπο και όχι από υπολογιστή. Οι πιο διαδεδομένες τους μορφές είναι είτε παραμορφωμένες εικόνες κειμένου, είτε εικόνες που συναντάμε στην καθημερινότητα (εικόνα 3.24). Η συμπλήρωσή του Captcha είναι απαραίτητη στη πλειοψηφία των διαδικτυακών εφαρμογών και έχει ως σκοπό να σταματήσει τις αυτοματοποιημένες ενέργειες που χρησιμοποιούνται για την κακόβουλη είσοδο σε αυτές. Το κύριο χαρακτηριστικό τους είναι ότι παρουσιάζουν ένα πρόβλημα, η επίλυση του οποίου είναι εύκολη για τους ανθρώπους αλλά πολύ δύσκολη για τους υπολογιστές (internet bots).

Ο συνηθέστερος τρόπος χρήσης του συστήματος Captcha, είναι μέσω έτοιμων λύσεων, καθώς η δημιουργία ενός τέτοιου συστήματος είναι αρκετά δύσκολη και πολύπλοκη.



Εικόνα 3.24: Απεικόνιση Captcha εικόνας.

Η JCaptcha κλάση της Java παρέχει ένα ασφαλή μηχανισμό πιστοποίησης με τη χρήση μιας γεννήτριας εικόνων. Η JCaptcha μπορεί να χρησιμοποιηθεί σε μια διαδικτυακή εφαρμογή με διάφορους τρόπους (π.χ. με τη χρήση ενός servlet). Στο Wicket αν θέλεις να κάνεις χρήση του JCaptcha μπορείς να το κάνεις με το servlet, αλλά είναι προτιμότερο να χρησιμοποιηθούν τα resource του ίδιου του framework. Παρακάτω θα παρουσιαστεί ένα μέρος του component από την υλοποίηση του JCapctcha σε μια φόρμα Wicket εφαρμογής.

Αρχικά χρειαζόμαστε ένα πεδίο κειμένου (TextField) και μια φόρμα (Form). Για να είναι επιτυχής η ταυτοποίηση (validation) θα πρέπει τα δεδομένα που εισήχθησαν στο πεδίο κειμένου να είναι ίδια με αυτά που εμφανίζονται στην Captcha απεικόνιση. Με τη μέθοδο getImageCaptchaservice() παίρνουμε κάθε φορά τις εικόνες για την Captcha υλοποίηση. Σε κάθε λανθασμένη είσοδο του χρήστη στο πεδίο κειμένου, που σημαίνει ότι πληκτρολόγησε κάτι διαφορετικό σε σχέση με αυτό που εμφανίζεται στην εικόνα, καλείται η μέθοδος onError() (εικόνα 3.25).

```
public abstract class TextFieldCaptchaInput extends TextField {
    public TextFieldCaptchaInput (String id, IModel model,
                                  String challengeId) {
    super(id, model);
    add(new AbstractValidator() {
        @Override
        protected void onValidate(IValidatable validatable) {
            if (!getImageCaptchaService().validateResponseForID(
                challengeId, validatable.getValue())) {
                onError(this, validatable);
            }
        }
    });
    }
}
protected abstract CaptchaImageService getImageCaptchaService();

@Override
protected void onComponentTag(final ComponentTag tag) {
    super.onComponentTag(tag);
    tag.put("value", "");
}

protected abstract void onError(
    AbstractValidator validator, IValidatable validatable);
}
```

1.Έλεγχος εισόδου στο πεδίο

2. Καλείται για να φέρει εικόνες

3. onError μέθοδος

Εικόνα 3.25: JCaptcha component, για έλεγχο των δεδομένων του χρήστη

Αν ο χρήστης πληκτρολογήσει το κείμενο που εμφανίζεται στη εικόνα η ταυτοποίηση θα ολοκληρωθεί και θα οδηγηθεί στην επόμενη σελίδα της εφαρμογής.

3.5.4 Authorization strategies

Μια από τις βασικές λειτουργίες σε μια web εφαρμογή είναι η ταυτοποίηση του χρήστη (authentication). Κάθε φορά που ο χρήστης εισέρχεται στην εφαρμογή, για να καταφέρει να εκτελέσει κάποια λειτουργία της, πρέπει υποχρεωτικά να πληκτρολογήσει όνομα χρήστη και κωδικό πρόσβασης. Τι γίνεται όμως στη

περίπτωση που θέλουμε να ελέγχουμε την πρόσβαση των χρηστών στα διάφορα σημεία της εφαρμογής; Θέλουμε για παράδειγμα, μια σελίδα στην οποία θα έχει πρόσβαση μόνο ο διαχειριστής (administrator) της εφαρμογής. Ο έλεγχος για το ποιος χρήστης θα έχει πρόσβαση, σε ποιο κομμάτι της εφαρμογής ονομάζεται εξουσιοδότηση (authorization). Παρακάτω παρουσιάζεται η χρήση του authorization strategy στην ebook διαδικτυακή εφαρμογή.

Ο στόχος είναι η δημιουργία μια σελίδας στην οποία θα έχει πρόσβαση μόνο ο διαχειριστής και μέσα από αυτή θα μπορεί να διαγράψει οποιονδήποτε από τους απλούς χρήστες της εφαρμογής. Με την χρήση του annotation “AuthorizeInstantiation” δηλώνουμε ότι στη σελίδα UserManagerPage θα έχει πρόσβαση μόνο ο διαχειριστής. (εικόνα 3.26).

```
@AuthorizeInstantiation("admin")
public class UserManagerPage extends ElibraryPage {
    private static final long serialVersionUID = 1L;

...
```

Εικόνα 3.26: annotation για πρόσβαση στη σελίδα μόνο από τον διαχειριστή.

Δημιουργούμε μια κλάση η οποία υλοποιεί την διεπαφή IRoleCheckingStrategy η οποία με τη σειρά της ελέγχει αν ο χρήστης που έχει αποθηκευτεί στο session είναι ο διαχειριστής (εικόνα 3.27).

```
public class UserRolesAuthorizer implements IRoleCheckingStrategy {

    @Override
    public boolean hasAnyRole(Roles roles) {
        ElibrarySession authSession = (ElibrarySession) Session.get();
        try {
            return authSession.getUserLogin().hasAnyRole(roles);
        }
```

Εικόνα 3.27: Κλάση ελέγχου της “ομάδας” που ανήκει ο χρήστης.

Στη κλάση UserLogin, η οποία κρατά τα στοιχεία του χρήστη, προσθέτουμε μια επιπλέον ιδιότητα τύπου “Role”. Κάθε φορά που ο χρήστης εισάγεται επιτυχημένα στην εφαρμογή μέσω της session κλάσης, αποθηκεύεται στη Role ιδιότητα ο χρήστης (εικόνα 3.28).

```
...
public boolean isAuthenticated() {
    if (userLogin != null) {
        Roles roles = new Roles(getUserLogin().getUsername());
    }
    return false;
}
...

public class UserLogin implements Serializable {
    private static final long serialVersionUID = 1L;

    ...

    private Roles roles;


    /**
     * Whether this user has any of the given roles.
     *
     * @param roles
     *            set of roles
     * @return whether this user has any of the given roles
     */
    public boolean hasAnyRole(Roles roles) {
        return this.roles.hasAnyRole(roles);
    }

    ...

}
```

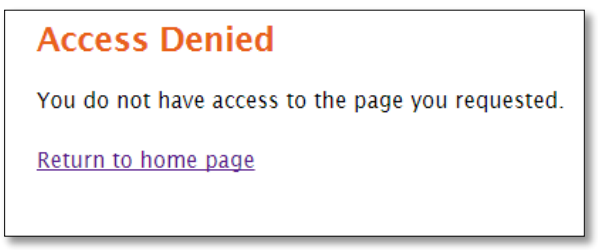
Εικόνα 3.28: Αποθήκευση στη Role του ονόματος του χρήστη.

Στην Application κλάση και πιο συγκεκριμένα στη μέθοδο init() δηλώνουμε ότι κάθε φορά που η εφαρμογή ξεκινά (bootstrapping), ενεργοποιείται η authorization strategy (εικόνα 3.29).

```
public class ElibraryApplication extends WebApplication {

    ...

    @Override
    public void init() {
        getDebugSettings().setDevelopmentUtilitiesEnabled(true);
        getSecuritySettings().setAuthorizationStrategy(
        new RoleAuthorizationStrategy(new UserRolesAuthorizer()));
        MetaDataRoleAuthorizationStrategy.authorize(UserManagerPage.class, "admin");
    }

    ...
```

Εικόνα 3.29: Authorization strategy κατά την εκκίνηση της εφαρμογής.

Αν ο χρήστης που συνδέεται επιτυχώς με την εφαρμογή έχει username “admin” τότε έχει πρόσβαση στην UserManagerPage, σε αντίθετη περίπτωση (δηλαδή κάποιος χρήστης με τα βασικά δικαιώματα) θα του εμφανιστεί το παρακάτω μήνυμα (εικόνα 3.30).



Εικόνα 3.30: Μήνυμα μη δικαιώματος πρόσβασης στη σελίδα.

3.6 Behaviors

Τα components μπορούν να εμπλουτιστούν με behaviors για να επεκτείνουν τις λειτουργίες τους. Στα component μπορείς να προσθέσεις ένα behavior υλοποιώντας κάποια/ες από τις μεθόδους που παρέχει η abstract κλάση “Behavior” (εικόνα 3.31).

<abstract> Behavior
beforeRender(Component) : void afterRender(Component) : void bind(Component) : void detach(Component) : void exception(Component, RuntimeException) : void getStatelessHint() : boolean isEnabled(Component) : boolean onComponentTag(Component, ComponentTag) : void isTemporary() : boolean

Εικόνα 3.31: Μέθοδοι Behavior κλάσης.

Όλες οι μέθοδοι εκτός από την isTemporary έχουν σαν κοινό στοιχείο ότι παίρνουν σαν παράμετρο ένα component. Με αυτόν τον τρόπο εξασφαλίζεται το stateless των behavior, καθώς δεν κρατιέται κάποια αναφορά στο component, αλλά συνδέεται ολόκληρο στο behavior.

Τα behaviors χρησιμοποιούνται κυρίως για δυο λόγους:

- Για να τροποποιήσουν τις ιδιότητες και τα χαρακτηριστικά των HTML tags.
- Για να αποκριθούν σε γεγονότα (events) ή κλήσεις (calls) που συμβαίνουν, στα components με τα οποία είναι συνδεδεμένα.

Για την πρώτη περίπτωση, υποθέτουμε ότι έχουμε ένα κείμενο το οποίο περικλείεται από div tag (εικόνα 3.32).

```
<div wicket:id="tagBehavior"></div>
```

Εικόνα 3.32: HTML κώδικας για div tag.

Στο Java αρχείο, αρχικά δημιουργούμε το Label component με το wicket id “tagBehavior” και σαν δεύτερη παράμετρο το κείμενο που θα εμφανιστεί στην οθόνη. Στη περίπτωση που θέλουμε το κείμενο που θα εμφανιστεί στην εφαρμογή να έχει σαν background το κόκκινο χρώμα, υλοποιούμε τη μέθοδο onComponentTag με τον παρακάτω κώδικα (εικόνα 3.33).

```
Label tagBehavior = new Label("tagBehavior", "Red background text");

tagBehavior.add(new Behavior() {

    @Override
    public void onComponentTag(Component component, ComponentTag tag) {
        // TODO Auto-generated method stub
        tag.put("style", "background-color:red");               }
});
queue(tagBehavior);
```

Εικόνα 3.33: Behavior για αλλαγή background χρώματος.

Για την δεύτερη περίπτωση, τα behaviors τα οποία θέλουν να λαμβάνουν κλήσεις από τα components, πρέπει να υλοποιήσουν το interface IBehaviorListener. Τέτοιες κλήσεις μπορεί να είναι για παράδειγμα ίδιου τύπου με αυτές που λαμβάνει ένα link, αλλά πλέον το behavior τις λαμβάνει μέσω του component με το οποίο είναι συνδεδεμένο.

Η AJAX είναι η κυριότερη τεχνολογία που υποστηρίζει το Wicket ώστε τα behaviors να αποκρίνονται στα διάφορα γεγονότα που συμβαίνουν στα component. Τα AJAX behavior κληρονομούν την AbstractAjaxBehavior, η οποία με τη σειρά της

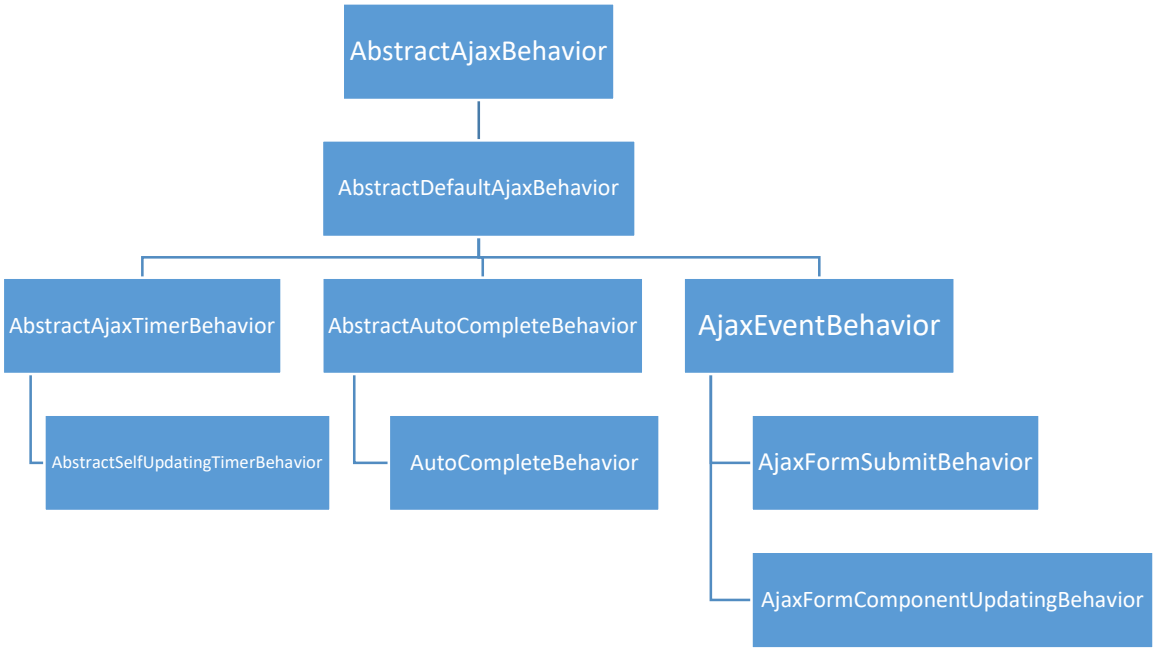
Η AJAX (Asynhronous Javascript And Xml) τεχνολογία συνδυάζει υπάρχουσες τεχνολογίες (Javascript, XML, HTML, DOM, CSS) ώστε να καταστήσει την επικοινωνία client με server πιο άμεση και τις σελίδες που το χρησιμοποιούν πιο ζωντανές. Το κύριο χαρακτηριστικό μιας web σελίδας που χρησιμοποιεί AJAX, είναι η άμεση ενημέρωσή της με νέο περιεχόμενο χωρίς να χρειάζεται να ξαναφορτωθεί εξ' ολοκλήρου.

κάνει "extend" την Behavior κλάση και υλοποιεί τις μεθόδους της διεπαφής IBehaviorListener (διάγραμμα 3.10). Στο επόμενο παράδειγμα (εικόνα 3.34) δημιουργούμε δύο Textfield components. Στο δευτερο Textfield προσθέτουμε το AJAX behavior "AjaxFormComponentUpdatingBehavior".

```
TextField<String> field1 = new TextField<String>("field1");
TextField<String> field2 = new TextField<String>("field2")
.add(new AjaxFormComponentUpdatingBehavior("onkeyup") {
    @Override
    protected void onUpdate (final AjaxRequestTarget target) {
        field1.setModelObject(null);
        target.add(field1);
    }
});
```

Εικόνα 3.34: Παράδειγμα AJAX behavior υλοποίησης.

Με το συγκεκριμένο behavior δηλώνουμε ότι κάθε φορά που ο χρήστης απελευθερώνει το κουμπί του πληκτρολογίου, ενώ βρίσκεται στο δεύτερο TextField, το περιεχόμενο στο πρώτο πεδίο σβήνεται.



Διάγραμμα 3.10: Ιεραρχία Ajax behavior κλάσεων.

Κεφάλαιο 4
Προδιαγραφές εφαρμογής ηλεκτρονικής βιβλιοθήκης

4.1 Εισαγωγή

Με τον όρο ηλεκτρονική βιβλιοθήκη χαρακτηρίζουμε ένα σύνολο από πόρους και τεχνικές δυνατότητες για την δημιουργία, αναζήτηση και χρήση αυτών. Αποτελεί ένα σύστημα αποθήκευσης και ανάκτησης πληροφοριών το οποίο διαχειρίζεται ψηφιακά δεδομένα οποιασδήποτε μορφής.

Ηλεκτρονική βιβλιοθήκη χαρακτηρίζεται κάθε βιβλιοθήκη που παρέχει όλο της το υλικό σε ψηφιοποιημένη μορφή. Η ηλεκτρονική βιβλιοθήκη λειτουργεί μέσω διαδικτύου παρέχοντας στο χρήστη εύκολη και γρήγορη πρόσβαση από οποιοδήποτε σημείο. Κάθε ηλεκτρονική βιβλιοθήκη είναι συνδεδεμένη με μια βάση δεδομένων, στην οποία είναι αποθηκευμένα τα ψηφιακά βιβλία που αναζητά ο χρήστης.

4.2 Λειτουργικές απαιτήσεις

Με τον όρο λειτουργικές απαιτήσεις, δηλώνουμε τις υπηρεσίες που θα πρέπει να παρέχει το σύστημα και το πώς θα πρέπει να αντιδρά σε συγκεκριμένες καταστάσεις.

Στα πλαίσια της παρούσας διπλωματικής εργασίας, προχωρήσαμε στην υλοποίηση μιας εφαρμογής ηλεκτρονικής βιβλιοθήκης υιοθετώντας της αρχές και τα χαρακτηριστικά που διέπουν το εργαλείου ανάπτυξης web εφαρμογών Apache Wicket και διάφορων άλλων τεχνολογιών που θα περιγράφουν σε επόμενο κεφάλαιο. Στη συνέχεια παρουσιάζονται οι λειτουργίες που πρέπει να εκτελεί η εφαρμογή ηλεκτρονικής βιβλιοθήκης σύμφωνα με τις προδιαγραφές.

4.2.1 Διαδικασία εγγραφής χρήστη

Ο χρήστης για να ολοκληρώσει την εγγραφή του στην εφαρμογή πρέπει να συμπληρώσει επιτυχώς όλα τα πεδία της φόρμας.

- Για κάθε πεδίο κειμένου απαιτείται συγκεκριμένος συνδυασμός χαρακτήρων για τον οποίο ο χρήστης ενημερώνεται πατώντας το κουμπί πληροφοριών

δίπλα σε αυτό. Ο χρήστης πατώντας το κουμπί επιβεβαίωσης, σε περίπτωση μη έγκυρης συμπλήρωσης σε ένα η περισσότερα πεδία, θα ενημερωθεί με τα αντίστοιχα μηνύματα στο κάτω μέρος των λανθασμένων πεδίων.

- Τα πεδία κειμένου email και username είναι μοναδικά για κάθε χρήστη. Αν ο χρήστης συμπληρώσει κάποιο από τα παραπάνω πεδία με τιμή η οποία είναι ήδη καταχωρημένη στη βάση δεδομένων, πατώντας το κουμπί επιβεβαίωσης θα ενημερωθεί με το κατάλληλο μήνυμα λανθασμένης συμπλήρωσης.

4.2.2 Διαδικασία ταυτοποίησης χρήστη

Για να καταφέρει ο χρήστης να περιηγηθεί στις κύριες λειτουργίες της εφαρμογής πρέπει πρώτα να ολοκληρωθεί η ταυτοποίηση του. Σε περίπτωση λανθασμένης συμπλήρωσης, είτε του username είτε του password, ο χρήστης ενημερώνεται με το κατάλληλο μήνυμα και για τις δυο περιπτώσεις.

4.2.3 Διαδικασία αναζήτησης βιβλίου

Ο χρήστης έχει τη δυνατότητα να αναζητήσει ένα βιβλίο με τους εξής τρόπους:

- Συμπληρώνοντας το πεδίο κειμένου και επιλέγοντας κάποιο από τα διαθέσιμα κριτήρια αναζήτησης της λίστας (τίτλος, συγγραφέας, εκδότης, isbn13).
- Έχοντας τη δυνατότητα να επιλέξει ανάμεσα από δεκαέξι κατηγορίες βιβλίων πατώντας πάνω στα αντίστοιχα link.

Η αναζήτηση των βιβλίων για τις προαναφερθείσες περιπτώσεις, πραγματοποιείται στη βάση δεδομένων της εφαρμογής βάση των κριτηρίων που όρισε ο χρήστης. Αν η αναζήτηση στη βάση δεδομένων είναι επιτυχής τα αποτελέσματα εμφανίζονται σε ένα πίνακα με τις κυριότερες πληροφορίες των βιβλίων.

4.2.4 Διαδικασία download και βαθμολόγησης βιβλίου

Ο χρήστης μπορεί να δει αναλυτικά τις πληροφορίες του βιβλίου που έχει επιλέξει και να εκτελέσει τις παρακάτω λειτουργίες:

- Βαθμολόγηση βιβλίου στη κλίμακα από ένα έως πέντε. Κάθε φορά που ο χρήστης βαθμολογεί το βιβλίο, αυτομάτως ο μέσος όρος βαθμολογίας όλων των χρηστών εμφανίζεται στο αντίστοιχο πεδίο. Κάθε φορά που ο χρήστης

βαθμολογεί ένα βιβλίο, στη βάση δεδομένων της εφαρμογής αποθηκεύεται πάντα η τελευταία του βαθμολόγηση.

- Πατώντας το κουμπί “download”, το αρχείο αποθηκεύεται στον υπολογιστή του χρήστη.

4.2.5 Διαδικασία upload βιβλίου

Ο χρήστης έχει τη δυνατότητα να ανεβάσει στην εφαρμογή οποιοδήποτε βιβλίο σε μορφή “pdf” που δεν ξεπερνά σε μέγεθος τα 40 mb.

- Σε κάθε πεδίο κείμενου απαιτείται συγκεκριμένος συνδυασμός χαρακτήρων, για τον οποίο ο χρήστης ενημερώνεται πατώντας το κουμπί πληροφοριών δίπλα σε αυτό. Με το πάτημα του κουμπιού “upload” και σε περίπτωση μη έγκυρης συμπλήρωσης ενός πεδίου, ο χρήστης θα ενημερωθεί με το ανάλογο μήνυμα στο κάτω μέρος του λανθασμένου πεδίου.
- Για να αποφευχθεί η διπλή καταχώρηση κάποιου βιβλίου, στη βάση δεδομένων της εφαρμογής, το πεδίο isbn13 ελέγχεται ώστε να είναι μοναδικό. Αν ο χρήστης συμπληρώσει το συγκεκριμένο πεδίο με τιμή η οποία είναι ήδη καταχωρημένη στη βάση δεδομένων, πατώντας το κουμπί επιβεβαίωσης θα ενημερωθεί με το κατάλληλο μήνυμα.

4.2.6 Διαδικασία διαχείρισης χρηστών

Η συγκεκριμένη λειτουργία αφορά μόνο τον διαχειριστή της εφαρμογής. Τα στοιχεία όλων των χρηστών, ανακτώνται από τη βάση δεδομένων και εμφανίζονται σε έναν πίνακα. Ο διαχειριστής έχει τη δυνατότητα να διαγράψει οποιονδήποτε χρήστη της εφαρμογής με το πάτημα του link “remove”.

4.3 Μη λειτουργικές απαιτήσεις

Με τον όρο μη λειτουργικές απαιτήσεις χαρακτηρίζουμε τους περιορισμούς στις υπηρεσίες ή τις λειτουργίες που προσφέρει η εφαρμογή.

Οι μη λειτουργικές απαιτήσεις περιγράφουν το πώς ή το πόσο καλά το σύστημα θα υποστηρίξει τις λειτουργικές απαιτήσεις της εφαρμογής. Υπό άλλη έννοια ορίζουν τους λόγους σύμφωνα με τους οποίους θα θεωρηθεί το σύστημα επιτυχημένο. Στη συνέχεια παρουσιάζονται οι κυριότερες μη λειτουργικές απαιτήσεις της ηλεκτρονικής βιβλιοθήκης.

4.3.1 Απαιτήσεις υλοποίησης

Για την κατασκευή της ηλεκτρονικής βιβλιοθήκης θα χρησιμοποιηθεί το εργαλείο Apache Wicket λόγω της δυνατότητας ανάπτυξης εφαρμογών με χαρακτηριστικά όπως ο διαχωρισμός εμφάνισης και περιεχομένου η επαναχρησιμοποίηση του κώδικα και η ασφάλεια.

Θα γίνει χρήση της Guice dependency injection τεχνολογίας, με το κυριότερο πλεονέκτημά της να είναι η μείωση του επαναλαμβανόμενου κώδικα στα αντικείμενα της εφαρμογής. Ολόκληρη τη διαδικασία της αρχικοποίησης ή του καθορισμού των εξαρτήσεων τη χειρίζεται μια εξωτερική οντότητα η οποία συντονίζει κάθε αντικείμενο στο όλο σύστημα.

Επιπλέον είναι χρήσιμη στο unit testing, επειδή είναι εύκολο να εμβάλει ένα προσχέδιο (mock) της υπηρεσίας, μέσα στο αντικείμενο που τεστάρετε, αλλάζοντας το αρχείο ρυθμίσεων ή τα δεδομένα των μεταβλητών στον χρόνο εκτέλεσης.

4.3.2 Απαιτήσεις χρηστικότητας

Η εφαρμογή θα πρέπει να είναι όσο το δυνατόν πιο απλή και φιλική προς τον χρήστη. Ο επισκέπτης θα πρέπει να μπορεί να πηγαίνει άμεσα και εύκολα στις πληροφορίες που ψάχνει. Η φιλικότητα προς τον χρήστη (user friendly) είναι από τους σημαντικότερους παράγοντες κατάταξης της εφαρμογής στις μηχανές αναζήτησης. Ένα επιπλέον κριτήριο φιλικότητας, στο οποίο οι μηχανές αναζήτησης δίνουν ιδιαίτερη σημασία, είναι η προσαρμογή της εφαρμογής σε οθόνες κινητών και tablets. Για την υλοποίηση των παραπάνω απαιτήσεων, στην εφαρμογή ηλεκτρονικής βιβλιοθήκης, επιλέχτηκε το εργαλείο BootStrap.

4.3.3 Απαιτήσεις ιδιωτικότητας

Στη σημερινή εποχή, ο όγκος των δεδομένων που καθημερινά διακινείται μέσω του διαδικτύου είναι τεράστιος. Επειδή είναι σχεδόν αδύνατο να ελεγχθεί το σύνολο των πληροφορίων ενός ατόμου, θα πρέπει να δημιουργούνται ζώνες ιδιωτικότητας οι οποίες θα προστατεύουν αυτά τα δεδομένα.

Ένα μέτρο που ενισχύει την ιδιωτικότητα, είναι η κρυπτογράφηση των κωδικών πρόσβασης των χρηστών. Είναι πολύ κακή πρακτική να αποθηκεύονται οι κωδικοί πρόσβασης, απευθείας χωρίς καμία επεξεργασία στη βάση δεδομένων της εφαρμογής. Αν η βάση δεδομένων κλαπεί, λόγω παραβίασης των συστημάτων, τότε ο θύτης αποκτά πρόσβαση σε όλους τους κωδικούς. Για να αποφευχθούν

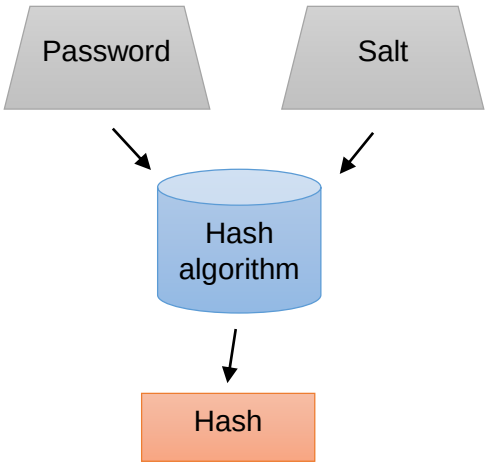
τέτοια δυσάρεστα γεγονότα μια αποτελεσματική τεχνική είναι η κρυπτογράφηση με τη τεχνική “Hashing και Salting” [http://hasging&salting].

Για τη δημιουργία ενός λογαριασμού, στην εφαρμογή ηλεκτρονικής βιβλιοθήκης, ακολουθούνται τα επόμενα βήματα (εικόνα 4.1):

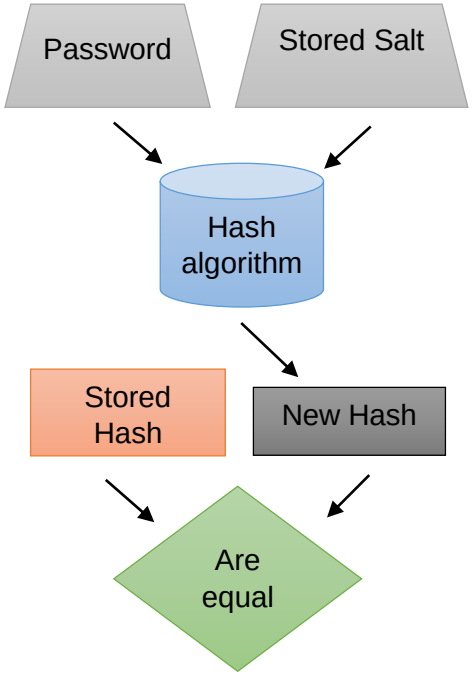
- Δημιουργία μιας τυχαίας συμβολοσειράς Salt και αποθήκευση αυτής στη βάση δεδομένων.
- Στον κωδικό που έχει εισάγει ο χρήστης προστίθεται η Salt συμβολοσειρά (Salt + Password).
- Στα προηγούμενα δεδομένα εφαρμόζεται ένας αλγόριθμος κατατεμαχισμού (hash algorithm) και η παραγόμενη hash συμβολοσειρά αποθηκεύεται στη βάση δεδομένων.

Για να συνδεθεί ο χρήστης στην εφαρμογή ακολουθείται η παρακάτω διαδικασία (εικόνα 4.2).

- Ο χρήσης εισάγει των κωδικό του.
- Σε αυτόν τον κωδικό προστίθεται η αποθηκευμένη Salt συμβολοσειρά.
- Στα νέα δεδομένα εφαρμόζεται ο αλγόριθμος κατακερματισμού. Αν η παραγόμενη hash συμβολοσειρά συμπίπτει με την αποθηκευμένη, τότε η ταυτοποίηση του χρήστη είναι επιτυχής. Σε αντίθετη περίπτωση ο χρήστης δεν μπορεί να συνδεθεί με την εφαρμογή.



Εικόνα 4.1: Δημιουργία κωδικού.

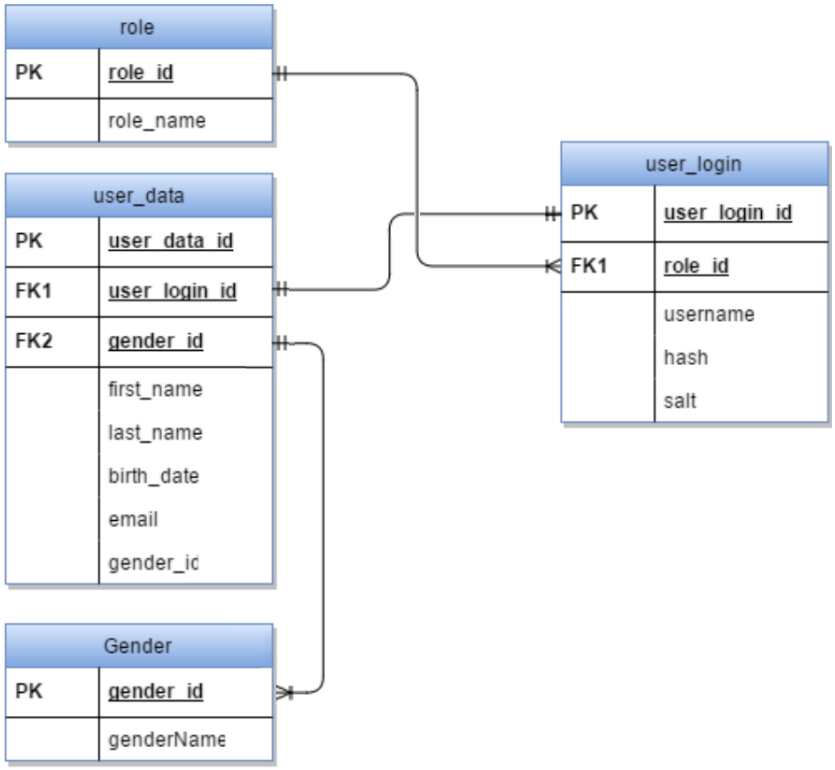


Εικόνα 4.2: Ταυτοποίηση κωδικού.

Με τη παραπάνω διαδικασία εξασφαλίζεται ότι ο μόνος που γνωρίζει των κωδικό ενός λογαριασμού και κατά συνέπεια μπορεί να συνδεθεί στην εφαρμογή, είναι ο κάτοχος του λογαριασμού.

4.3.4 Απαιτήσεις Βάσεων δεδομένων

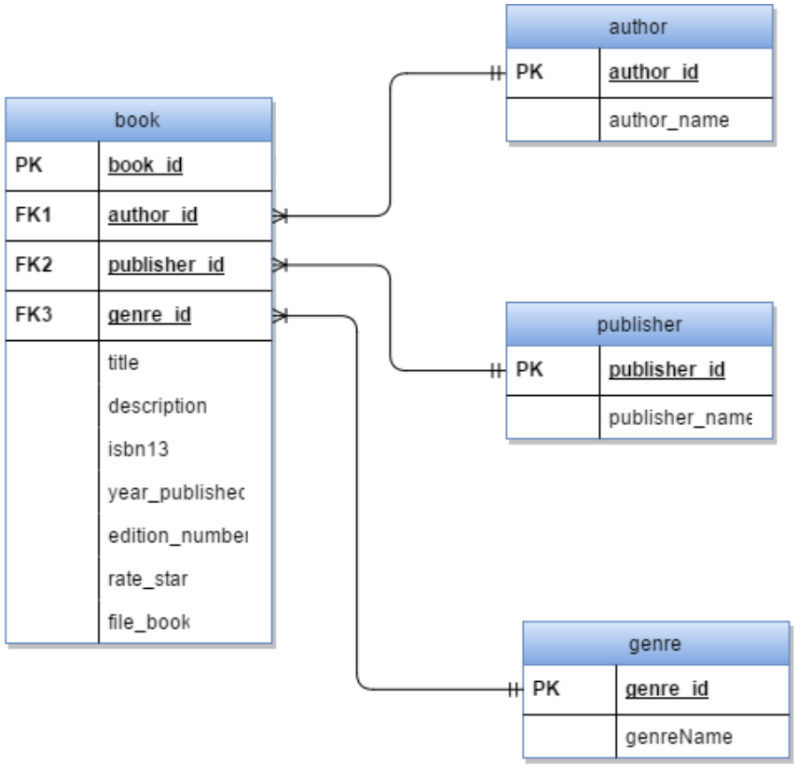
Το σύστημα διαχείρισης βάσεων δεδομένων που επιλέχθηκε για την υλοποίηση της ηλεκτρονικής βιβλιοθήκης είναι η PostgreSQL. Στη συνέχεια παρουσιάζονται οι πίνακες και οι συσχετίσεις τους, για τη βάση δεδομένων της εφαρμογής.



Εικόνα 4.3: Συσχετίσεις πινάκων διαχείρισης χρηστών.

- **role:** Στον πίνακα role δηλώνονται οι ομάδες χρηστών, ανάλογα με τα δικαιώματά τους. Στη παρούσα υλοποίηση εκτός του διαχειριστή (administrator) όλοι οι υπόλοιποι είναι απλοί εγγεγραμμένοι χρήστες (registered).
- **user_login:** Στον πίνακα user_login αποθηκεύονται όλα τα απαραίτητα δεδομένα που αφορούν την ταυτοποίηση των χρηστών με την εφαρμογή.
- **user_data:** Σε αυτό το πίνακα κρατούνται επιπλέον πληροφορίες, που αφορούν τους εγγεγραμμένους χρήστες.
- **gender:** Στον πίνακα gender δηλώνονται οι ιδιότητες των χρηστών (άνδρας, γυναίκα).

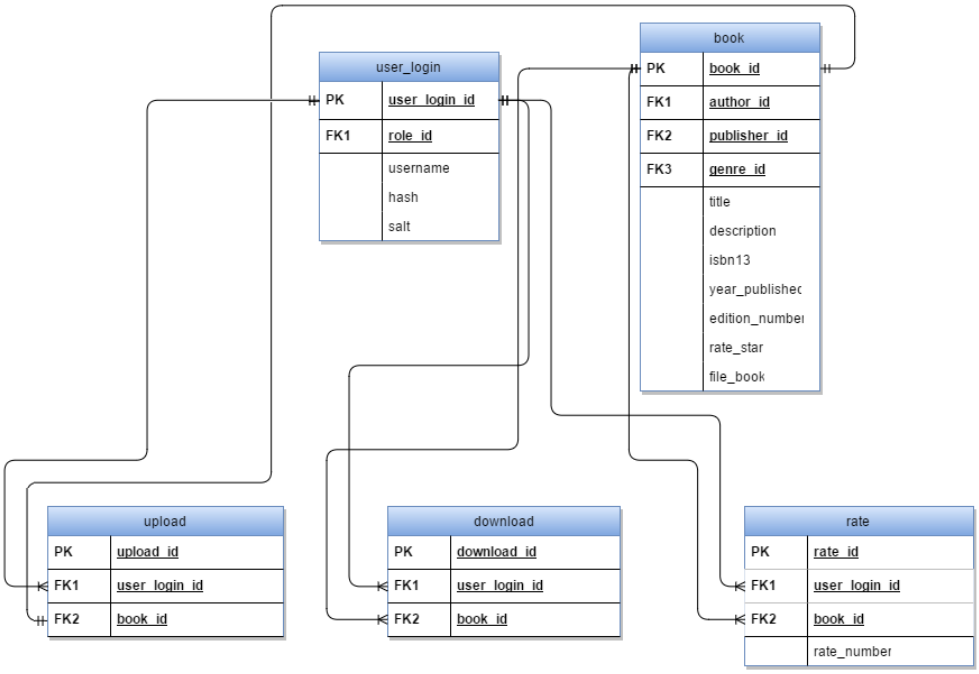
Η σχέση του user_login με τον role πίνακα είναι ένα προς πολλά (one to many) διότι ένας χρήστης μπορεί να ανήκει μόνο σε μία ομάδα χρηστών. Τα στοιχεία ταυτοποίησης του χρήστη με τις υπόλοιπες προσωπικές πληροφορίες του, έχουν συσχέτιση ένα προς ένα (one to one).



Εικόνα 4.4: Συσχετίσεις πινάκων διαχείρισης βιβλίων.

- **book:** Στον πίνακα αποθηκεύονται όλες οι πληροφορίες που αφορούν τα ψηφιοποιημένα βιβλίο.
- **author:** Πίνακας με τα στοιχεία των συγγραφέων.
- **publisher:** Πίνακας με τα στοιχεία των εκδοτών.
- **genre:** Στον πίνακα genre περιέχονται οι κατηγορίες των βιβλίων.

Η σχέση του book με τον author πίνακα είναι πολλά προς ένα (many to one). Αυτό οφείλεται στο γεγονός πως ένα βιβλίο έχει γραφτεί από ένα συγγραφέα, ο ίδιος όμως συγγραφέας μπορεί να έχει γράψει πολλά βιβλία. Οι ίδιοι συσχετισμοί ισχύουν για τους πίνακες publisher και genre.



Εικόνα 4.5: Συσχετίσεις πινάκων διαχείρισης στατιστικών.

- **rate:** Στον πίνακα rate αποθηκεύονται οι βαθμολογίες των χρηστών για τα ψηφιοποιημένα βιβλία**.**
- **download:** Σε αυτό τον πίνακα δηλώνεται ο αριθμός των download κάθε βιβλίου.
- **upload:** Στον upload πίνακα κρατούνται, για στατιστικούς λόγους, τα στοιχεία του βιβλίου και του χρήστη που το ανέβασε.

Η σχέση των user_login και book με τον rate πίνακα είναι ένα προς πολλά (one to many). Ο συνδυασμός των ξένων κλειδιών, user_login_id και book_id, είναι μοναδικός (unique), που σημαίνει ότι ένας χρήστης μπορεί να βαθμολογήσει μόνο μια φορά ένα βιβλίο. Σε περίπτωση που θελήσει να το βαθμολογήσει ξανά, τότε ο πίνακας ενημερώνεται με τη νεότερη βαθμολογία. Σε αντίθεση με τον πίνακα download όπου ο χρήστης μπορεί να κατεβάσει όσες φορές θέλει ένα βιβλίο, στον upload ο χρήστης έχει τη δυνατότητα να ανεβάσει μόνο κάποιο βιβλίο που δεν υπάρχει ήδη στη βάση.

4.3.5 Απαιτήσεις επικοινωνίας με διαφορετικά συστήματα

Είναι απαραίτητο ένα σύστημα να μπορεί να επικοινωνεί με άλλα συστήματα ώστε είτε να παίρνει τις πληροφορίες που χρειάζεται, είτε να παράγει δεδομένα για αυτά. Όταν υπάρχουν απαιτήσεις για επικοινωνία μιας εφαρμογής με άλλα συστήματα, θα πρέπει να προσδιορίζεται με ακρίβεια τόσο ο τρόπος της

επικοινωνίας (π.χ. πρωτόκολλο επικοινωνίας, φυσική σύνδεση), όσο και οι πληροφορίες που θα ανταλλάσσονται.

Η επικοινωνίας της ηλεκτρονικής βιβλιοθήκης με το σύστημα διαχείρισης βάσεων δεδομένων postgreSQL, θα πραγματοποιηθεί με χρήση της τεχνολογίας Hibernate. Ο όγκος των δεδομένων μιας εφαρμογής φυλάσσεται σε μια συγκεκριμένη πηγή αποθήκευσης. Συχνά υπάρχει η ανάγκη να μεταφερθούν αυτά τα δεδομένα από ένα σύστημα διαχείρισης βάσεων δεδομένων σε ένα άλλο. Το κενό αυτό έρχεται να καλύψει το Hibernate με τη δυνατότητα να συνδέεται σε βάσης δεδομένων διαφορετικών κατασκευαστών χωρίς να τροποποιείται ο κώδικας της εφαρμογής παρά μόνο κάποιες παράμετροι σε ένα ειδικό αρχείο.

Κεφάλαιο 5
Τεχνολογίες

5.1 BootStrap

Το BootStrap [http://getbootstrap.com/] είναι μια συλλογή εργαλείων ανοιχτού κώδικα (ελεύθερο λογισμικό) για τη δημιουργία ιστοσελίδων και διαδικτυακών εφαρμογών. Για την υλοποίηση της διεπαφής της εφαρμογής έγινε χρήση του BootStrap έκδοσης 3.3.6 , το οποίο παρέχει υποστήριξη RWD (Responsive Web Design) [https://en.wikipedia.org/wiki/Responsive_web_design] για παρουσίαση της εφαρμογής σε διάφορες οθόνες συσκευών.

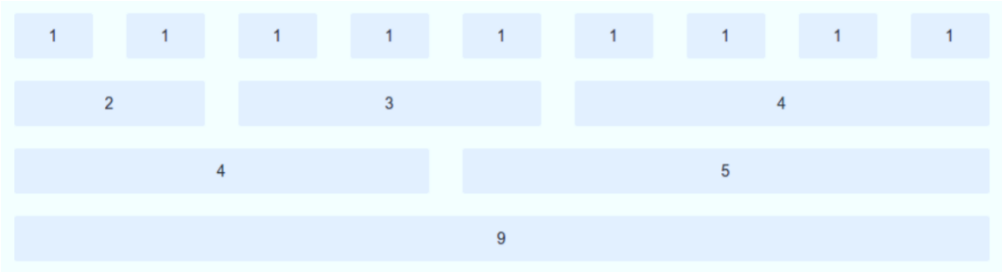
Το Responsive Web Design (RWD) είναι η τεχνολογία που επιτρέπει σε μια ιστοσελίδα να προσαρμόζεται αυτόματα στην εκάστοτε συσκευή. Είτε ο χρήστης μπει από τον υπολογιστή, είτε από το κινητό ή το tablet του, η ιστοσελίδα θα προσαρμοστεί ανάλογα με το μέγεθος της οθόνης.

Το BootStrap είναι το πιο δημοφιλές framework ανάπτυξης RWD εφαρμογών, ενώ περιέχει πέραν των στοιχείων CSS και στοιχεία HTML και JavaScript. Το BootStrap υποστηρίζει δύο γνωστούς προεπεξεργαστές (Preprocessors), τους LESS και SASS. Οι προεπεξεργαστές είναι γλώσσες που ερμηνεύονται σε μορφή CSS. Η SASS σχεδιάστηκε για να απλοποιήσει και να επεκτείνει τη CSS, ενώ η LESS ήθελε να κρατήσει τη συντακτική ομοιότητα με τη CSS. Η χρήση του BootStrap σε μια εφαρμογή είναι σχετικά απλή, το μόνο που απαιτείται είναι η δήλωση των εξωτερικών αρχείων CSS στην κεφαλίδα του HTML της κύριας σελίδας. Αν ο προγραμματιστής θέλει να χρησιμοποιήσει στοιχεία JavaScript και JQuery, θα πρέπει και αυτά με τη σειρά τους να δηλωθούν στο HTML έγγραφο (εικόνα 5.1).

```
<html>
<head>
<title>elibrary</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="css/bootstrap-theme.min.css">
<link rel="stylesheet" href="css/bootstrap-datepicker3.min.css">
<script src="js/bootstrap.min.js"></script>
<script src="js/bootstrap-datepicker.min.js"></script>
<link rel="icon" href="img/etab.png" type="image/x-icon" />
</head>
<body>

...

</body>
</html>
```

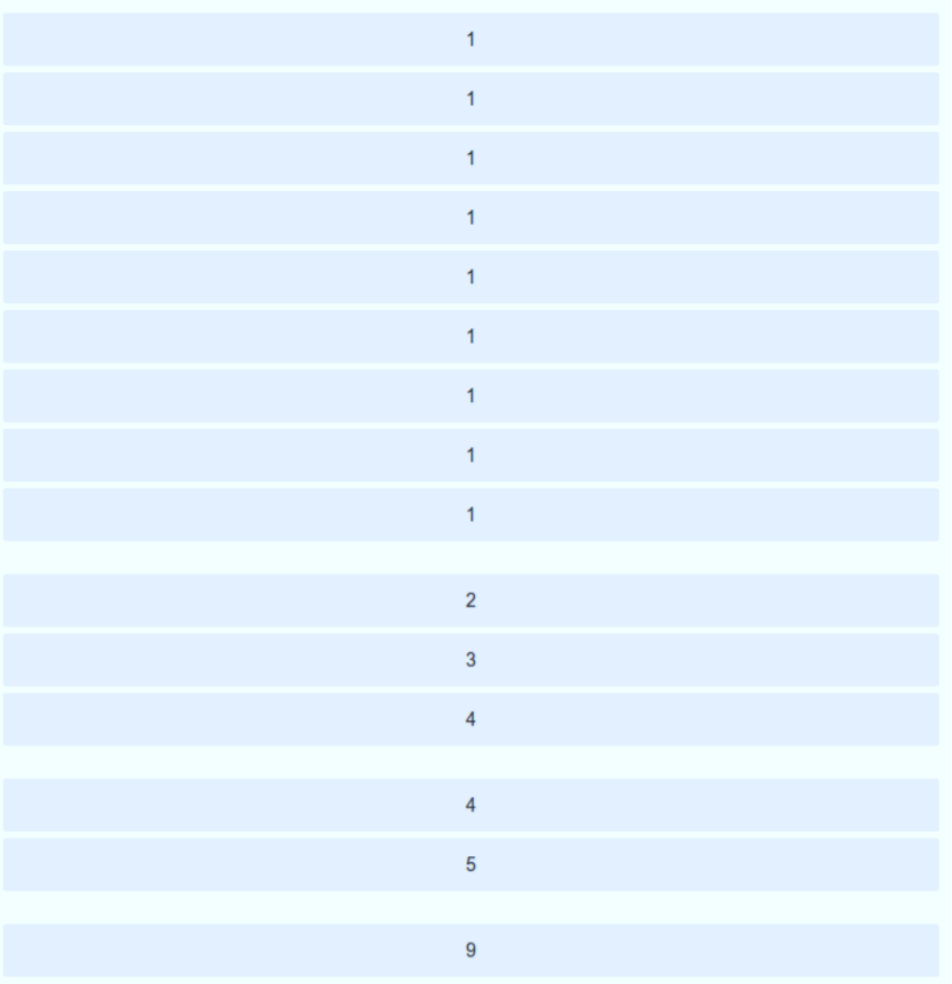
Εικόνα 5.1: Δήλωση Bootstrap αρχείων στη κεφαλίδα του HTML.

Το σύστημα πλέγματος του BootStrap χρησιμοποιεί 12 στήλες, καλύπτοντας μια περιοχή 940pixel, χωρίς ενεργά responsive χαρακτηριστικά. Αν προστεθεί στην κεφαλίδα το κατάλληλο αρχείο CSS, το πλέγμα προσαρμόζεται μέχρι τα 724pixel μήκος επί 1170pixel πλάτος, ανάλογα με την ανάλυση της οθόνης (εικόνα 5.2).



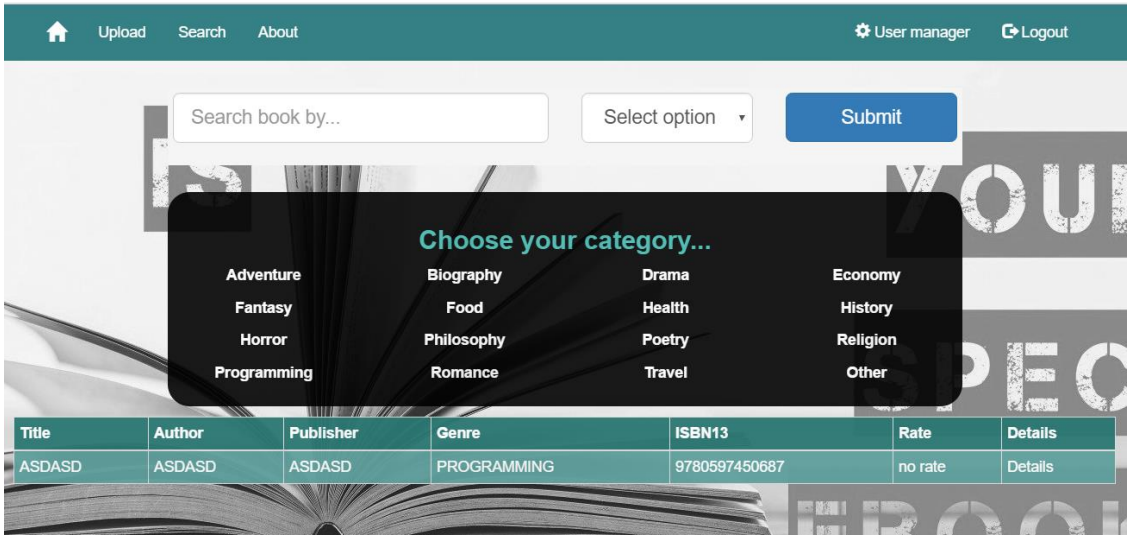
Εικόνα 5.2: Σύστημα πλέγματος για οθόνες μεγαλύτερες των 724pixel.

Σε οθόνη κάτω των 767pixel, οι στήλες “σπάνε” και στοιχίζονται κατακόρυφα (εικόνα 5.3).



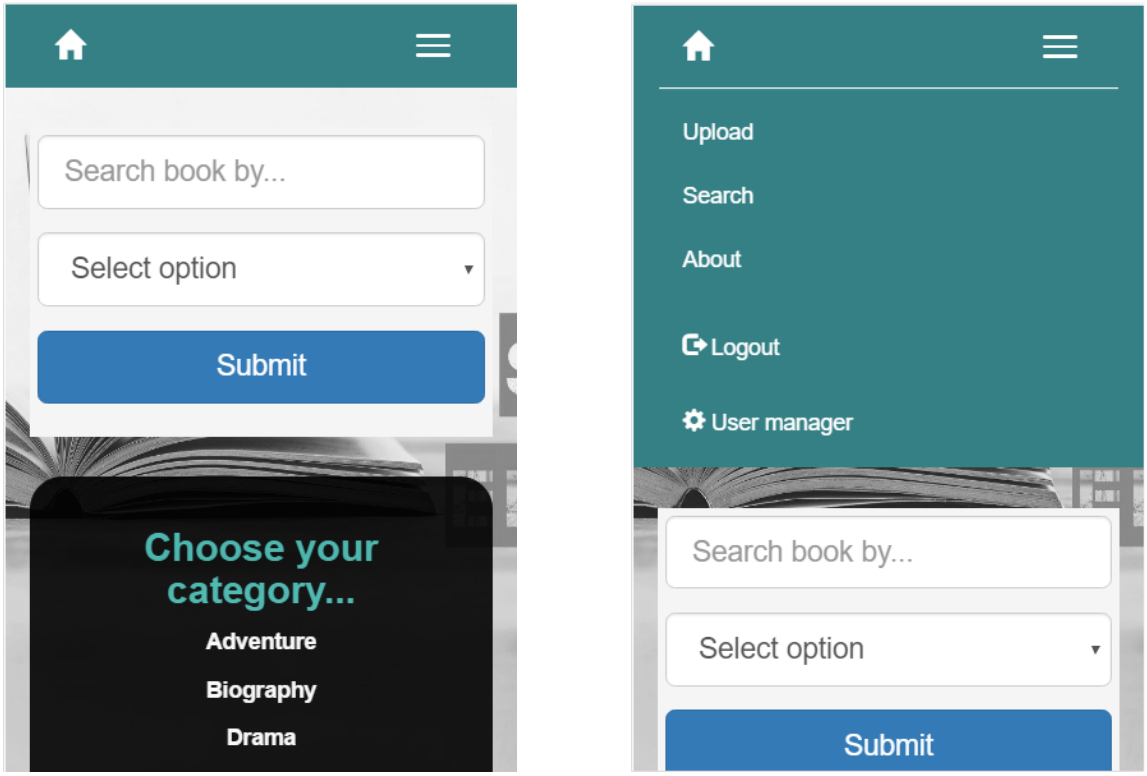
Εικόνα 5.3: Σύστημα πλέγματος για οθόνες μικρότερες των 724 pixels.

Στην εικόνα 5.4 φαίνεται η σελίδα αναζήτησης της ηλεκτρονικής βιβλιοθήκης σε οθόνη με ανάλυση 1920x1080.



Εικόνα 5.4: Σελίδα αναζήτησης σε οθόνη 1920x1080.

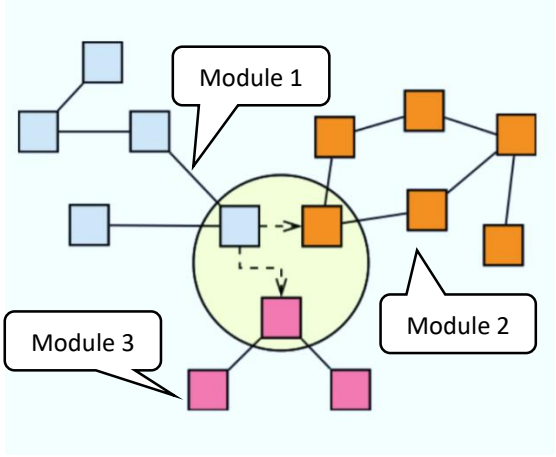
Η ίδια σελίδα, αυτή τη φορά, σε κινητή συσκευή (smartphone) με ανάλυση οθόνης 360x640.



Εικόνα 5.5: Σελίδα αναζήτησης σε οθόνη smartphone.

5.2 Guice

Στη κατασκευή της ηλεκτρονικής βιβλιοθήκης χρησιμοποιήθηκε το Guice έκδοση 5.0 της Google [http:///google-guice-dependency-injection], ένα ελαφρύ πλαίσιο που υλοποιεί το dependency injection πρότυπο για την γλώσσα προγραμματισμού Java. Μια εφαρμογή που ακολουθεί αυτό το πρότυπο αποτελείται από διάφορα modules. Στα modules περιέχονται όλες οι πληροφορίες παραμετροποίησης των συνδέσεων μεταξύ διεπαφών και υλοποιήσεων (εικόνα 5.6).



Εικόνα 5.6: Τα modules και οι συνδέσεις τους σε μια εφαρμογή.

Στην ηλεκτρονική βιβλιοθήκη έχει υλοποιηθεί το UserService interface με τη μέθοδο isValidUser (εικόνα 5.7).

```
public interface UserService {

    UserLogin isValidUser(String username, String password);

}
```

Εικόνα 5.7: Διεπαφή UserService.

Αντίστοιχα έχει δημιουργηθεί η κλάση UserServiceImpl, η οποία αποτελεί υλοποίηση της παραπάνω διεπαφής (εικόνα 5.8)

```
public class UserServiceImpl implements UserService {

    public UserLogin isValidUser(String username, String password) {

    ...

    }
}
```

Εικόνα 5.8: Κλάση UserServiceImpl που υλοποιεί την UserService.

Στο ServiceModule (εικόνα 5.9) γίνετε σύνδεση (bind) της διεπαφής UserService με την υλοποίησή της UserServiceImpl.

```
import com.google.inject.AbstractModule;

public class ServiceModule extends AbstractModule {

    @Override
    protected void configure() {
        bind(UserService.class).to(UserServiceImpl.class);
        bind(BookService.class).to(BookServiceImpl.class);
    }
}
```

Εικόνα 5.9: Module που συνδέει τις διεπαφές με τις υλοποιήσεις.

Η κλάση η οποία είναι υπεύθυνη για την κατασκευή των αντικειμένων είναι ο Injector. Για την αρχικοποίηση του Injector χρειάζονται τα modules. Όταν ζητηθεί η κατασκευή ενός αντικειμένου συγκεκριμένου τύπου, αρχικά βρίσκει τι να κατασκευάσει αφού μπορεί να υπάρχουν πολλές υλοποιήσεις, στη συνέχεια λύνει τις εξαρτήσεις και τέλος αρχικοποιεί το αντικείμενο. Κατά την εκκίνηση της εφαρμογής, η παραμετροποίηση όσον αφορά τις υλοποιήσεις που θα χρησιμοποιηθούν, παρέχεται από τα modules.

Το annotation “@Inject”, θα χρησιμοποιηθεί από το Guice ώστε να κάνει “inject” την υλοποίηση του UserService. Στη συνέχεια δημιουργούμε έναν Injector αντικείμενο, του δίνουμε σαν παράμετρο το UserServiceModule και αρχικοποιούμε το UserService στιγμιότυπο. Τέλος είμαστε έτοιμοι να κάνουμε χρήση της μεθόδου isValidUser() (εικόνα 5.10).

```
@Inject
UserService userService;

Injector injector = Guice.createInjector(new UserServiceModule());
userService = injector.getInstance(UserService.class);

...

userService.isValidUser("George", "1234")

...
```

Εικόνα 5.10: Αρχικοποίηση Injector με UserServiceModule.

5.3 JPA

Στις περισσότερες εφαρμογές, χρησιμοποιείται κάποια σχεσιακή βάση δεδομένων που αποτελείται από πίνακες και δεδομένα που σχετίζονται μεταξύ τους. Στις επαγγελματικές εφαρμογές, στις οποίες απαιτείται η συντήρηση και η αποθήκευση πολλών δεδομένων, η πιο συνηθισμένη λύση για την εφαρμογή αυτών των λειτουργιών είναι η χρήση των DAO (Data Access Objects). Αυτά τα αντικείμενα λειτουργούν ως διαμεσολαβητές μεταξύ των αντικειμένων μια εφαρμογής (POJOS) και της βάσης που θα αποθηκεύσει τα δεδομένα των αντικειμένων.

Για την επικοινωνία μεταξύ των αντικειμένων μια εφαρμογής και της βάσης δεδομένων, δημιουργήθηκε η διασύνδεση προγραμματισμού εφαρμογών JPA (Java Persistence API) [http://docs.oracle.com/javaee/]. Η JPA έχει σαν στόχο την καθιέρωση ενός συγκεκριμένου μοντέλου αποθήκευσης δεδομένων, το οποίο να μπορεί να εφαρμοστεί σε κάθε είδος εφαρμογή της Java. Μια εφαρμογή που χρησιμοποιεί τις διασυνδέσεις της JPA είναι ανεξάρτητη από το framework που θα χρησιμοποιήσει.

Ένα από τα σημαντικότερα χαρακτηριστικά της JPA είναι ο τρόπος με τον οποίο καθορίζεται η εφαρμογή της σχεσιακής αντιστοίχισης των αντικειμένων (ORM). Η διαδικασία αποθήκευσης ενός αντικειμένου μπορεί να δηλωθεί είτε μέσω ενός ξεχωριστού αρχείου (xml) που έχει την ευθύνη της αντιστοίχισης (mapping) είτε μέσω annotation (στη περίπτωση της ηλεκτρονικής βιβλιοθήκης χρησιμοποιήθηκε η τεχνική του annotation) που τοποθετούνται απευθείας στην κλάση που μας ενδιαφέρει.

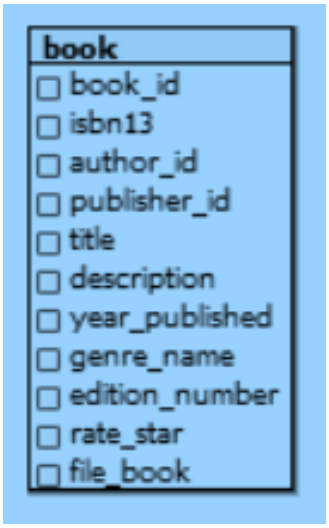
Ένα άλλο σημαντικό πλεονέκτημα της σχεσιακής αντιστοίχισης των αντικειμένων σε σχέση με την προσέγγιση των σχεσιακών βάσεων δεδομένων, είναι η μείωση της ποσότητας του κώδικα που χρειάζεται να γραφτεί για τη διαχείριση μίας βάσης δεδομένων.

Από την πλευρά της JPA μπορούμε να πούμε πως κάθε αντικείμενο σε μία εφαρμογή μπορεί να εξελιχθεί σε οντότητα (entity) βάσης δεδομένων. Θα πρέπει να γίνει ένας σαφής διαχωρισμός ανάμεσα στα στιγμιότυπα αντικείμενων και στις οντότητες. Ένα στιγμιότυπο ενός αντικειμένου μένει για κάποιο χρονικό διάστημα στη μνήμη του υπολογιστή, ενώ μια οντότητα είναι ένα αντικείμενο το οποίο υπάρχει για κάποιο διάστημα στην μνήμη του υπολογιστή, αλλά και μόνιμα αποθηκευμένο σε κάποια βάση δεδομένων. Μια οντότητα μπορούμε να την αποθηκεύσουμε, να την διαγράψουμε και να την επεξεργαστούμε όπως εμείς θέλουμε. Η αντιστοίχιση

ενός αντικειμένου με τον πίνακα της βάσης δεδομένων, γίνεται με τη χρήση μεταπληροφοριών (metadata).

Παρακάτω θα παρουσιαστεί ένα παράδειγμα, σχεσιακής αντιστοίχισης των αντικειμένων, της οντότητας Book με χρήση annotation όπως περιγράφεται στο πακέτο JPA έκδοση 2.1 (εικόνα 5.11).

```
@Entity
@Table(name = "book", schema = "public")
public class Book implements Serializable {
    private static final long serialVersionUID = 1L;

    public Book() {
        super();
    }

    @Id
    @GeneratedValue
    @Column(name = "book_id")
    private int bookID;

    @Column(name = "title")
    private String title;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private Author author;

    ...

}
```



Εικόνα 5.11: Σχεσιακή αντιστοίχιση κλάσης με πίνακα βάσης.

Την κλάση Book την χαρακτηρίζουμε ως οντότητα με το annotation “@Entity” και την αντιστοιχίζουμε με τον πίνακα book της βάσης δεδομένων. Στη συνέχεια ορίζουμε το πρωτεύον κλειδί χρησιμοποιώντας τα annotation “@ID” και “@GeneratedValue” για να δηλώσουμε ότι το κλειδί παίρνει αυτόματα τιμές από την βάση δεδομένων. Με το “@Column” δηλώνουμε ότι η bookID ιδιότητα της κλάσης συσχετίζεται με την book_id στήλη της βάσης. Με το “@ManyToOne” δηλώνουμε ότι η οντότητα κλάση Book έχει μια σχέση πολλά προς λίγα με την οντότητα κλάση Author. Τέλος με το “@JoinColumn” annotation δηλώνουμε ότι το πεδίο author_id αποτελεί ξένο κλειδί με αντιστοιχία στο πίνακα Author.

5.4 Hibernate - ORM

Η JPA καθορίζει τον τρόπο με τον οποίο μπορεί να χρησιμοποιηθεί η τεχνική της σχεσιακής αντιστοίχισης των αντικειμένων στα πλαίσια ανάπτυξης μία εφαρμογής

λογισμικού με χρήση της γλώσσας προγραμματισμού Java. Η διασύνδεση προγραμματισμού JPA, παρέχει την διεπαφή, και ο προγραμματιστής επιλέγει το πακέτο που θα υλοποιήσει αυτή τη διεπαφή. Η υλοποίηση της JPA διασύνδεσης ονομάζεται πάροχος (Persistence Providers). Οι πιο δημοφιλείς πάροχοι της αγοράς είναι οι Hibernate [http://hibernate.org/], OpenJPA [http://openjpa.apache.org/] και EclipseLink [http://www.eclipse.org/eclipselink/].

Για να καλυφθεί το κενό της άμεσης επικοινωνίας μεταξύ του σχεσιακού μοντέλου και του μοντέλου των αντικειμένων μιας εφαρμογής, το Hibernate χρησιμοποιεί τη σχεσιακή αντιστοίχιση των αντικειμένων (Object Relational Mapping), η οποία είναι μία τεχνική που χρησιμοποιείται για τη μετατροπή δεδομένων ανάμεσα σε διαφορετικούς τύπους συστημάτων, με τη χρήση αντικειμενοστραφών γλωσσών προγραμματισμού. Με πιο απλά λόγια ORM ονομάζεται, η διαδικασία αντιστοίχισης ανάμεσα σε αντικείμενα της Java και πίνακες δεδομένων. Με την εφαρμογή της συγκεκριμένης τεχνικής, προκύπτει μια εικονική βάση αντικειμένων η οποία μπορεί να χρησιμοποιηθεί απευθείας μέσα από τη γλώσσα προγραμματισμού που χρησιμοποιείται για την ανάπτυξη της εκάστοτε εφαρμογής.

Η βιβλιοθήκη Hibernate, αρχικά ξεκίνησε να αναπτύσσεται από μια ομάδα προγραμματιστών της Cirrus Technologies με συντονιστή τον Gavin King. Η βιβλιοθήκη προσφέρει το απαραίτητο πλαίσιο εργασίας για την αντιστοίχιση του μοντέλου μίας εφαρμογής με μία σχεσιακή βάση δεδομένων. Για την επίτευξη αυτού, δημιουργούνται αντιστοιχίες μεταξύ των εννοιών του αντικειμενοστραφούς προγραμματισμού, όπως οι συσχετίσεις και η κληρονομικότητα και των πινάκων μίας σχεσιακής βάσης. Με τη εφαρμογή των παραπάνω ο προγραμματιστής βλέπει τελικά μία αντικειμενοστραφή βάση δεδομένων, παρ' όλο που στην ουσία χρησιμοποιεί μία σχεσιακή. Η Hibernate παρέχει την HQL (Hibernate Query Language) [https://docs.jboss.org/hibernate/], η οποία είναι παρόμοια με την SQL, για την σύνταξη και εκτέλεση αιτημάτων προς τη βάση.

Τα σημαντικότερα πλεονεκτήματα της Hibernate υλοποίησης συνοψίζονται παρακάτω:

- Η Hibernate παρέχει μια σειρά από εργαλεία που επιτρέπουν την εύκολη αποτύπωση αντικειμενοστραφών εννοιών σε σχήματα βάσης ασχέτως του είδους της βάσης που χρησιμοποιείται. Επιτρέπει τη χρήση κληρονομικότητας και πολυμορφισμού στις συσχετιζόμενες κλάσεις και

παρέχει την HQL γλώσσα ερωτημάτων, η οποία επιτρέπει τη χρήση όλων των παραπάνω.

- Επιτρέπει τη μεταφορά των εφαρμογών μεταξύ σχεσιακών βάσεων δεδομένων. Η μόνη διαφοροποίηση στις περισσότερες των περιπτώσεων είναι η αλλαγή της διαλέκτου επικοινωνίας με την βάση.
- Μπορεί να βοηθήσει να βελτιωθεί η απόδοση της εφαρμογής. Η Hibernate δημιουργεί πολύ αποδοτικά ερωτήματα, πράγμα που διασφαλίζει την απόδοση σε πολλές περιπτώσεις.
- Ο προγραμματιστής με τη χρήση της Hibernate για την κατασκευή της εφαρμογής του, αυξάνει σε μεγάλο βαθμό την παραγωγικότητά του, επειδή ενδιαφέρεται μόνο για τα αντικείμενα, τα οποία αποθηκεύονται στη βάση και ανακτώνται από αυτήν με ελάχιστο κόπο.

Στην elibrary εφαρμογή έγινε χρήση της Hibernate έδοσης 4.3.11, σε συνδυασμό με την postgreSQL σχεσιακή βάση δεδομένων. Παρακάτω θα παρουσιαστούν τα βασικότερα βήματα που χρειάζονται για να επιτευχθεί μια σύνδεση μεταξύ της βάσης δεδομένων και της εφαρμογής με χρήση της Hibernate τεχνολογίας.

Βήμα 1ο

Το Hibernate χρησιμοποιεί το hibernate.cfg.xml αρχείο ώστε να πάρει όλες τις απαραίτητες πληροφορίες (properties) που απαιτούνται για τη σύνδεση με τη βάση δεδομένων. (εικόνα 5.12).

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
        <property name="hibernate.connection.username">postgres</property>
        <property name="hibernate.connection.password">*********</property>
        <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/elibraryDB</property>
        <property name="hibernate.default_schema">public</property>
        <property name="connection_pool_size">1</property>
        <property name="show_sql">true</property>
        <property name="hibernate.temp.use_jdbc_metadata_defaults">false</property>
        <mapping class="dskarpetis.elibrary.domain.Book" />
        <mapping class="dskarpetis.elibrary.domain.UserData" />

        ...

    </session-factory>
</hibernate-configuration>
```

Εικόνα 5.12: Δήλωση των απαραίτητων properties για σύνδεση με τη βάση.

Στο παραπάνω αρχείο γίνετε χρήση της βάσης "elibraryDB" με χρήστη “postgresql” και κωδικό πρόσβασης “*********”. Το χαρακτηριστικό στοιχεί είναι το tag dialect που έχει τιμή “org.hibernate.dialect.PostgreSQLDialect” και δηλώνει

στο Hibernate ότι θα χρησιμοποιήσει την postegreSQL βάση δεδομένων. Αυτό είναι και ένα από τα ισχυρά όπλα του Hibernate, ότι δηλαδή μπορεί να συνδεθεί σχεδόν με όλες τις βάσεις δεδομένων χωρίς να χρειάζεται καμία αλλαγή στην εφαρμογή μας παρά μόνο ένα xml tag. Επίσης στο ίδιο αρχείο θα πρέπει να δηλωθούν όλα τα entity objects της εφαρμογής στο configuration αρχείο.

Βήμα 2ο

Δημιουργούμε τους πίνακες στη βάση δεδομένων. Παρακάτω παρουσιάζεται ο sql κώδικας για τον user_data πίνακα (εικόνα 5.13).

```
CREATE TABLE public.user_data
(
  user_data_id integer NOT NULL DEFAULT nextval('auto_increment_userdata'::regclass),
  user_login_id integer,
  first_name character varying(50),
  last_name character varying(50),
  email character varying(50),
  birth_date date,
  gender_name character varying,
  CONSTRAINT pk_user_data_id PRIMARY KEY (user_data_id),
  CONSTRAINT fk_gender_name FOREIGN KEY (gender_name)
      REFERENCES public.gender (gender_name) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_user_login_id FOREIGN KEY (user_login_id)
      REFERENCES public.user_login (user_login_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT un_email UNIQUE (email),
  CONSTRAINT un_user_login_id UNIQUE (user_login_id)
)
```

Εικόνα 5.13: Πίνακας user_data.

Βήμα 3ο

Δημιουργούμε το POJO αντικείμενο UserData και κάνουμε την συσχέτιση (mapping) με τον πίνακα user_data, με χρήση annotation (εικόνα 5.14).

```
@Entity
@Table(name = "user_data", schema = "public")
public class UserData {
    public UserData() {
        super();
    }

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seq-gen3")
    @Column(name = "user_data_id")
    private int userDataID;

    @Column(name = "first_name")
    private String firstName;

    ...

    @ManyToOne
    @JoinColumn(name = "gender_name")
    private Gender gender;

    @OneToOne
    @JoinColumn(name = "user_login_id")
    private UserLogin userLogin;
}
```

Εικόνα 5.14: POJO κλάση με χρήση annotation για το mapping.

Βήμα 4ο

Δημιουργούμε μια κλάση, με όνομα HibernateUtil, στην οποία θα οριστούν όλες οι απαραίτητες διαδικασίες για την εγκαθίδρυση του Session με την βάση δεδομένων (εικόνα 5.15)

```
public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    private static ServiceRegistry serviceRegistry;
    static {
        try {
            Configuration configuration = new Configuration();
            configuration.configure();
            serviceRegistry = new StandardServiceRegistryBuilder()
            .applySettings(configuration.getProperties()).build();
            sessionFactory = configuration.buildSessionFactory(serviceRegistry);
        } catch (Throwable th) {
            System.err.println("Initial SessionFactory creation failed" + th);
            throw new ExceptionInInitializerError(th);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Εικόνα 5.15: Δημιουργία session με τη βάση δεδομένων.

Στην εικόνα 5.16 γίνεται επίδειξη της χρήσης της HQL γλώσσας με τη μέθοδο getUserBy (String username), η οποία ανάλογα με τα δεδομένα της παραμέτρου, ψάχνει μέσα στη βάση για το αν υπάρχει το συγκεκριμένο όνομα χρήστη. Αν η αναζήτηση είναι επιτυχής τότε επιστρέφει όλα τα δεδομένα του χρήστη σε αντίθετη περίπτωση επιστρέφει null (εικόνα 5.16).

```
public class UserDaoImpl implements UserDao {

...

    public UserLogin getUserBy(String username) {
        UserLogin userLogin;
        try {
            openSession();
            String hql = "from UserLogin where username=:username";
            Query query = session.createQuery(hql);
            query.setParameter("username", username);
            userLogin = (UserLogin) query.uniqueResult();
        } catch (Exception e) {
            if (tx != null)
                tx.rollback();
            e.printStackTrace();
            return null;
        } finally {
            session.disconnect();
        }
        return userLogin;
    }

...

}
```

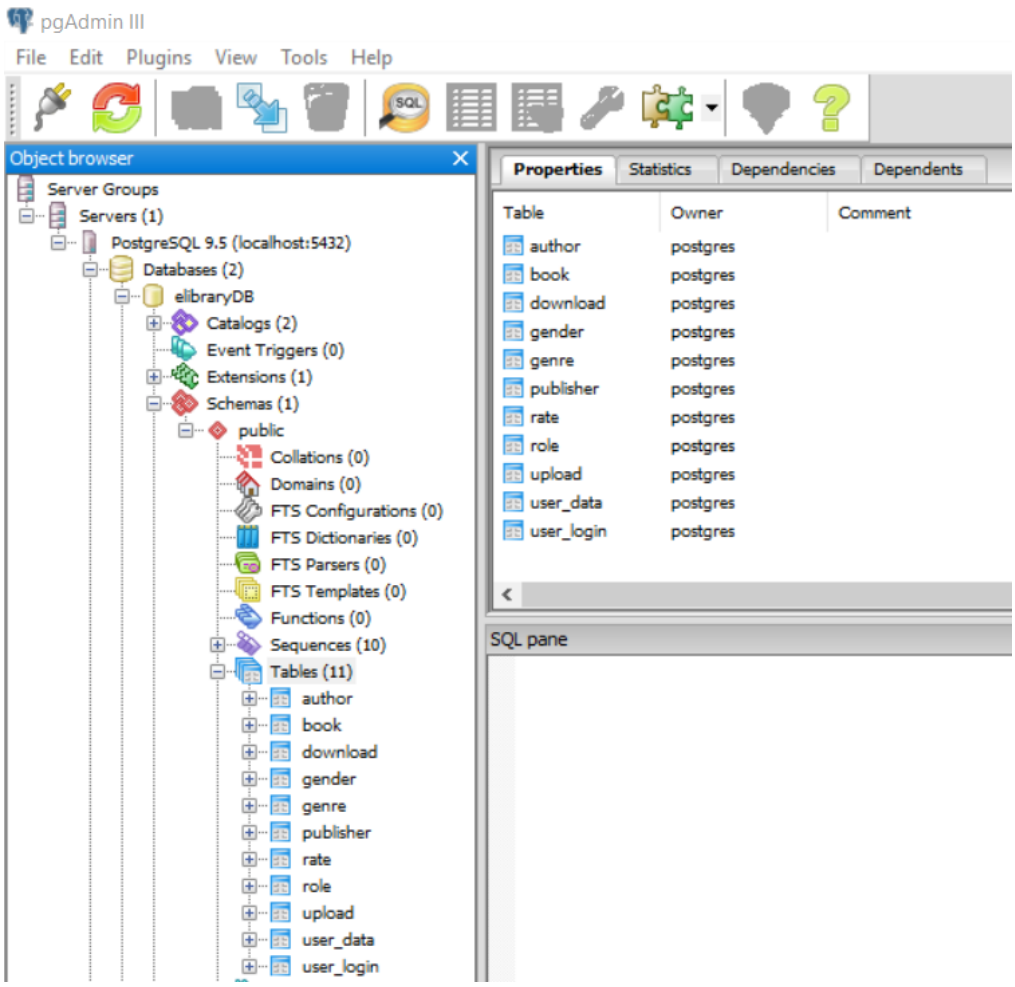
Εικόνα 5.16: Χρήση της HQL γλώσσας.

5.5 PostgreSQL

Η PostgreSQL [https://www.postgresql.org/] αποτελεί μια σχεσιακή βάση δεδομένων ανοιχτού κώδικα η οποία ξεκίνησε να αναπτύσσεται πριν από περίπου 20 χρόνια. Η αποδεδειγμένα καλή αρχιτεκτονική, η αξιοπιστία, η ακεραιότητα των δεδομένων και η ορθή λειτουργία της, την έχουν κατατάξει πολύ ψηλά στις επιλογές των προγραμματιστών.

Υποστηρίζει foreign keys, joins, views, triggers, και stored procedures. Συμπεριλαμβάνει τους περισσότερους τύπους δεδομένων όπως οι INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL και TIMESTAMP. Υποστηρίζει αποθήκευση binary large objects, όπως εικόνες και video. Επιπλέον διαθέτει native programming interfaces για Java, C/C++, .Net, Python, κ.α..

Η Διαχείριση της βάσης δεδομένων στην ηλεκτρονική βιβλιοθήκη γίνεται μέσω του εργαλείου pgAdmin έκδοση 1.22.1 [https://www.pgadmin.org/] (εικόνα 5.17).



Εικόνα 5.17: Διαχείριση της postgreSQL, με το πρόγραμμα pgAdmin.

Έχει σχεδιαστεί για να ικανοποιήσει τις ανάγκες των χρηστών, από απλό γράψιμο sql ερωτημάτων έως την ανάπτυξη πολύπλοκων βάσεων δεδομένων. Αρχικά είχε την ονομασία pgManager ενώ από το 1998 ξαναγράφηκε με νέα άδεια χρήσης με την σημερινή ονομασία pgAdmin. Είναι πλήρως γραμμένο στην γλώσσα προγραμματισμού C++ χρησιμοποιώντας τα wxWidgets ώστε να είναι συμβατό με τα κοινά λειτουργικά συστήματα όπως αναφέρθηκαν παραπάνω. Σήμερα είναι διαθέσιμο σε περισσότερες από δώδεκα γλώσσες.

Για την κατασκευή της βάσης δεδομένων χρησιμοποιήθηκε η postgreSQL εκδοση 9.4. Η βάση δεδομένων αποτελείται από έντεκα πίνακες οι οποίοι είναι: role, user_data, gender, user_login, book, author, publisher, genre, rate, download, upload (παράρτημα 1).

5.6 JUnit

Τα τεστ λογισμικού βοηθούν τον προγραμματιστή να βεβαιωθεί ότι η λογική του κώδικά του είναι σωστή. Στην εφαρμογή ηλεκτρονικής βιβλιοθήκης έγινε χρήση του JUnit έκδοση 5.12 [http://junit.org/junit4/], ενός από τα δημοφιλέστερα εργαλεία ελέγχου λογισμικού.

Το JUnit είναι μια βιβλιοθήκη της Java η οποία χρησιμοποιείται για τη δημιουργία unit testing. Το unit testing είναι μια διαδικασία ελέγχου ενός μικρού κομματιού κώδικα (π.χ. κλάσης, μεθόδου), για να βεβαιωθούμε ότι ανταποκρίνεται στις προδιαγραφές του.

Ένα unit test είναι μια κλάση που αποτελείται από διάφορες μεθόδους και η κάθε μια απ' αυτές αλληλοεπιδρά με το υπό δοκιμή κομμάτι κώδικα έτσι ώστε να βεβαιωθούμε ότι λειτουργεί σωστά.

5.7 Λοιπές τεχνολογίες

Apache Tomcat

Η Sun δημιούργησε τον πρώτο Servlet Container, έναν ειδικό Web Server ο οποίος ενσωματώνει ένα είδος Java εφαρμογών, τα Servlets. Ο Apache Software Foundation κατασκεύασε με τη σειρά του το Jserv Container.

Το 1999 τα δύο Servlet Container συνδυάστηκαν για τη δημιουργία του Tomcat application server. Στην ηλεκτρονική βιβλιοθήκη έγινε χρήση του Apache Tomcat έκδοση 8 [http://tomcat.apache.org/]. Για την διαχείριση των βασικών λειτουργιών

και εφαρμογών διατίθεται το πρόσθετο πρόγραμμα Tomcat Manager Web Application.

Ο application server αποτελείται από διάφορους φακέλους. Όλες οι βασικές ρυθμίσεις καθορίζονται από την επεξεργασία των αρχείων που βρίσκονται μέσα στο φάκελο conf. Η ανάπτυξη και οι ρυθμίσεις κάθε εφαρμογής γίνονται στον φάκελο WEB-INF. Μέσα σε αυτό περιέχεται το αρχείο web.xml, το οποίο περιέχει ρυθμίσεις για την λειτουργία της εφαρμογής.

Eclipse IDE

Το Eclipse IDE (Integrated Development Environment) [https://www.eclipse.org/] είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών. Χρησιμοποιείται για την ανάπτυξη εφαρμογών σε Java, ενώ με την βοήθεια διάφορων πρόσθετων που παρέχει, είναι δυνατή η ανάπτυξη εφαρμογών και σε άλλες γλώσσες.

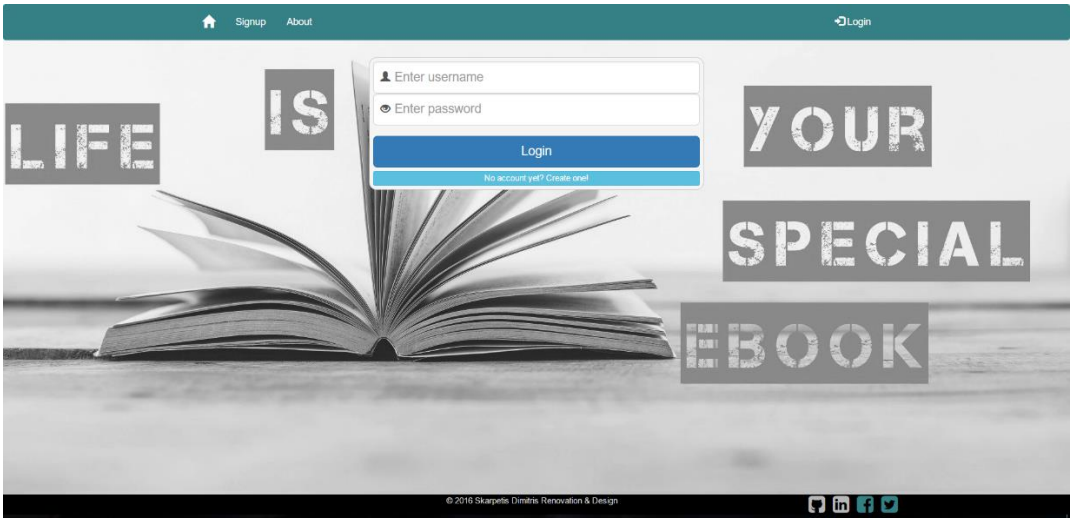
Για τη κατασκευή της ηλεκτρονικής βιβλιοθήκης έγινε χρήση του εργαλείου Eclipse Kepler. Το Eclipse αποτελεί ένα ιδανικό περιβάλλον για την οργάνωση του κώδικα μια ολόκληρης εφαρμογής. Λειτουργίες όπως, διατήρηση εκδόσεων και αλλαγές στον κώδικα μεταξύ των διαφόρων μελών της ομάδας ανάπτυξης, παρέχονται εντός του ίδιου του περιβάλλοντος.

Κεφάλαιο 6

Λειτουργίες εφαρμογής ηλεκτρονικής βιβλιοθήκης

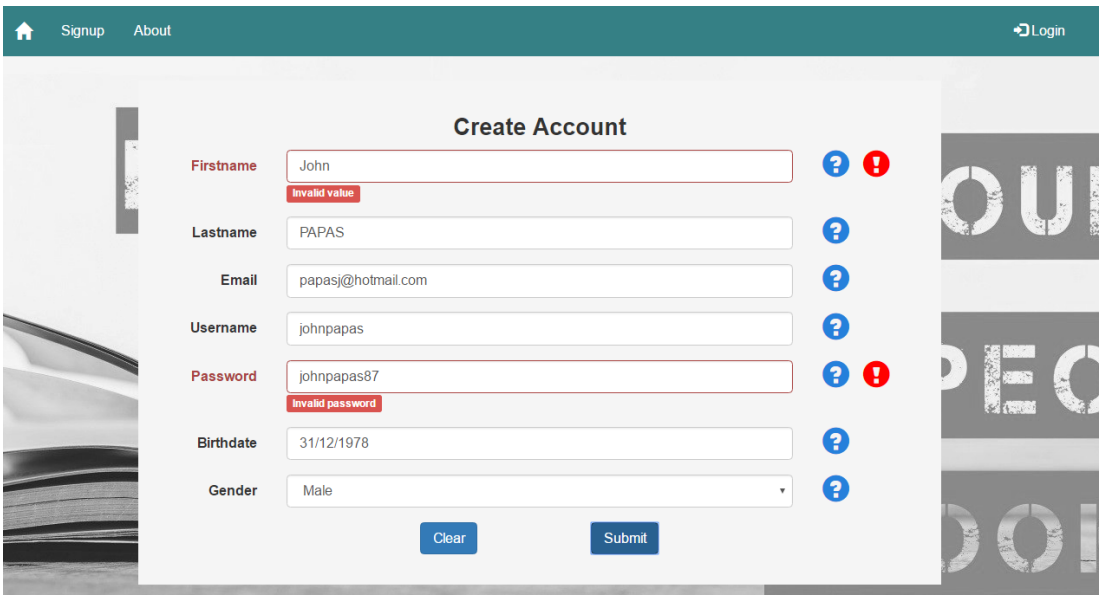
6.1 Εγγραφή χρήστη

Ο χρήστης μόλις επισκεφθεί την ηλεκτρονική βιβλιοθήκη, θα βρεθεί στην αρχική σελίδα της εφαρμογής (εικόνα 6.1).



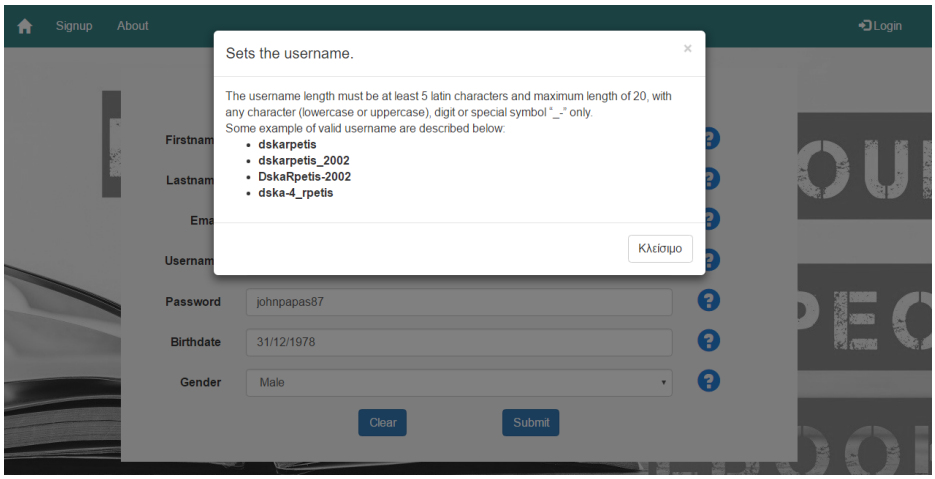
Εικόνα 6.1: Αρχική σελίδα εφαρμογής.

Ο χρήστης αν δεν έχει ήδη λογαριασμό στη εφαρμογή, μπορεί είτε πατώντας στο κουμπί “No account yet? Create one!” είτε στο “Signup” link στο navigation bar της σελίδας, να δημιουργήσει έναν (εικόνα 6.2).



Εικόνα 6.2: Εισαγωγή στοιχείων, για εγγραφή του χρήστη στην εφαρμογή.

Δίπλα από κάθε πεδίο βρίσκεται ένα κουμπί πληροφοριών, το οποίο βοηθά τον χρήστη στην ορθή εισαγωγή των δεδομένων (εικόνα 6.3).



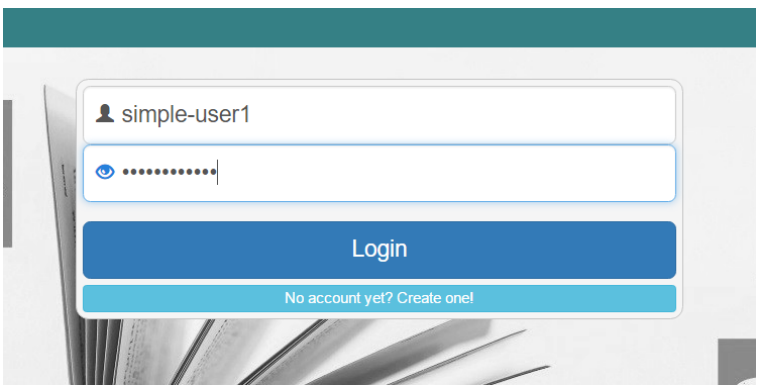
Εικόνα 6.3: Κουμπί πληροφοριών, ορθής συμπλήρωσης δεδομένων.

Όταν ο χρήστης πατήσει το “Submit” κουμπί ενεργοποιούνται οι server side validators της εφαρμογής. Σε περίπτωση που σε κάποιο πεδίο, εισαχθούν δεδομένα που δεν τηρούν τις προδιαγραφές της εφαρμογής τότε:

- Το Label (π.χ. Firstname) αλλάζει σε κόκκινο χρώμα.
- Το περίγραμμα του πεδίου κειμένου “κοκκινίζει” και εμφανίζεται λεκτικό που μας ενημερώνει για τα λανθασμένα δεδομένα.
- Εμφανίζεται ένα θαυμαστικό διπλά από το κουμπί πληροφορίων.

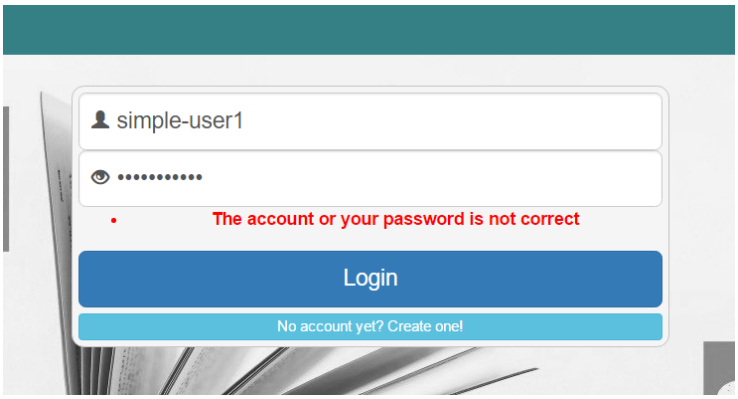
6.2 Ταυτοποίηση χρήστη

Αφού ο χρήστης ολοκληρώσει την εγγραφή του στην εφαρμογή, γυρνώντας στην αρχική σελίδα (με το πάτημα του “Home” εικονιδίου στο πάνω αριστερά μέρος της οθόνης), είναι έτοιμος να συμπληρώσει το όνομα χρήστη και τον κωδικό ώστε να γίνει η ταυτοποίηση (εικόνα 6.4).



Εικόνα 6.4: Συμπλήρωση στοιχείων για ταυτοποίηση χρήστη.

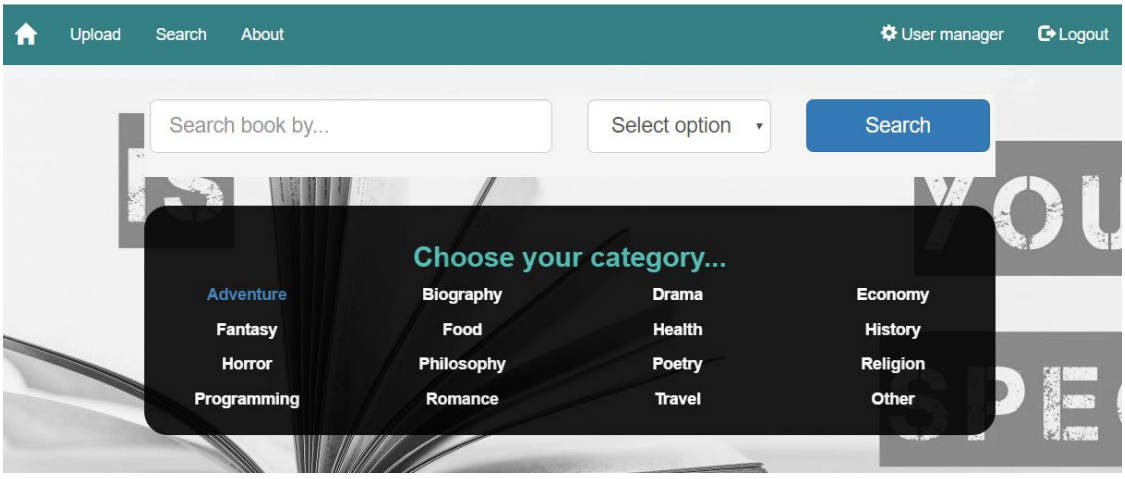
Σε περίπτωση λανθασμένης εισαγωγής είτε του ονόματος χρήστη είτε του κωδικού, ο χρήστης ενημερώνεται με το κατάλληλο μήνυμα (εικόνα 6.5).



Εικόνα 6.5: Λανθασμένη συμπλήρωση κωδικού χρήστη.

6.3 Αναζήτηση βιβλίου

Μετά την επιτυχημένη ταυτοποίηση ο χρήστης μεταφέρεται στη σελίδα αναζήτησης.

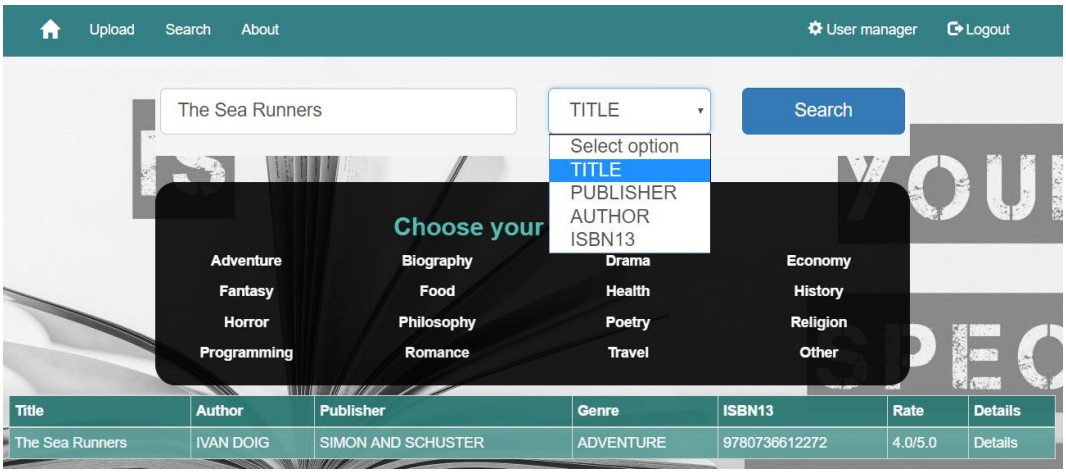


Εικόνα 6.6: Λανθασμένη συμπλήρωση κωδικού χρήστη.

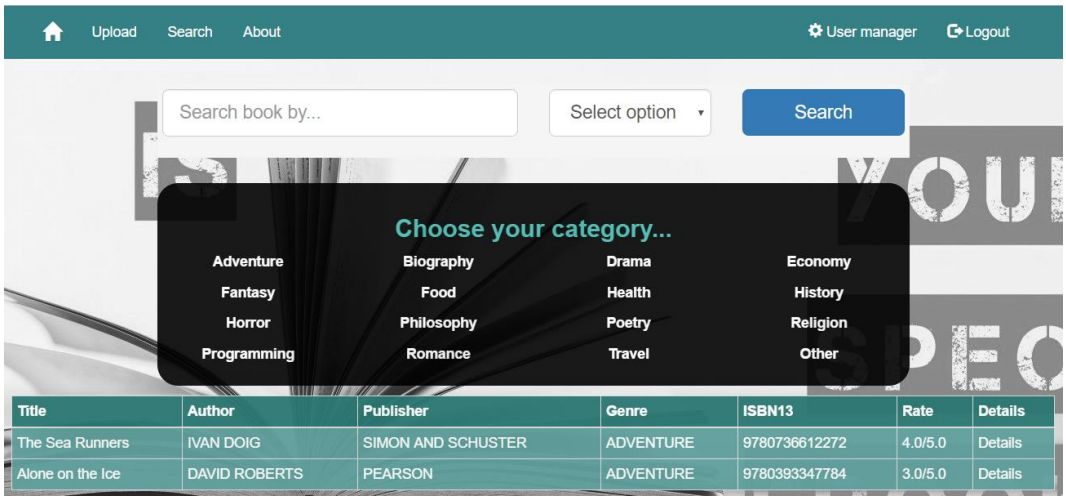
Ο χρήστης έχει τη δυνατότητα να αναζητήσει ένα βιβλίο με τους εξής τρόπους:

- Συμπληρώνοντας το πεδίο κειμένου, ο χρήστης έχει τη δυνατότητα να αναζητήσει ένα βιβλίο βάση τίτλου, συγγραφέα, εκδότη ή ISBN13 (εικόνα 6.7). Πατώντας το κουμπί “Search”, γίνεται η αναζήτηση βάση των κριτηρίων που τέθηκαν και αν το βιβλίο είναι διαθέσιμο τότε εμφανίζεται στην οθόνη του χρήστη.
- Ο χρήστης έχει τη δυνατότητα να επιλέξει ανάμεσα από δεκαέξι κατηγορίες βιβλίων. Πατώντας σε ένα από τα link, θα του εμφανιστούν όλα τα διαθέσιμα

βιβλία που υπάρχουν στη βάση δεδομένων της εφαρμογής για τη συγκεκριμένη κατηγορία (εικόνα 6.8).



Εικόνα 6.7: Αναζήτηση βάση κριτηρίων.



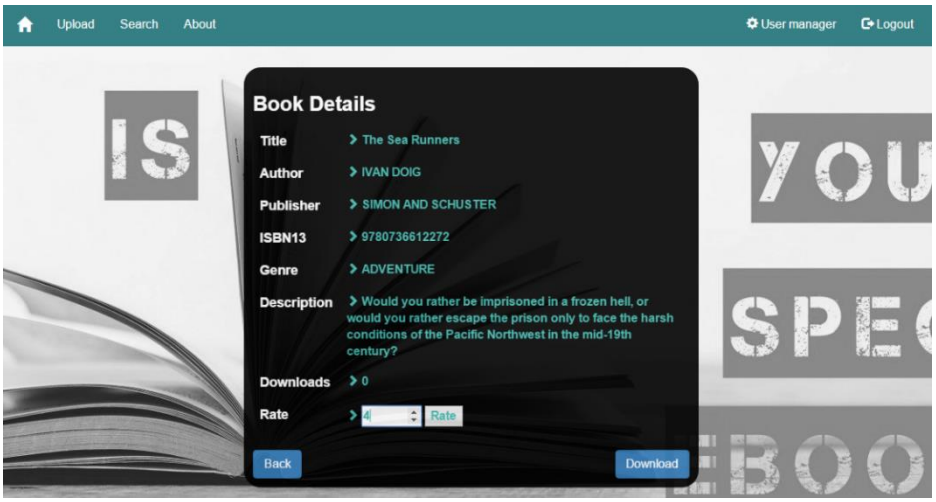
Εικόνα 6.8: Αναζήτηση βάση κατηγορίας.

6.4 Download και βαθμολόγηση βιβλίου

Στα αποτελέσματα της αναζήτησης ενός βιβλίου, παρατηρούμε την στήλη “Rate”. Σε αυτή τη στήλη φαίνεται ο μέσος όρος βαθμολόγησης των χρηστών, σε κλίμακα από 1 έως 5, για το συγκεκριμένο βιβλίο (εικόνα 6.9).

Εικόνα 6.9: Αναζήτηση βάση κατηγορίας.

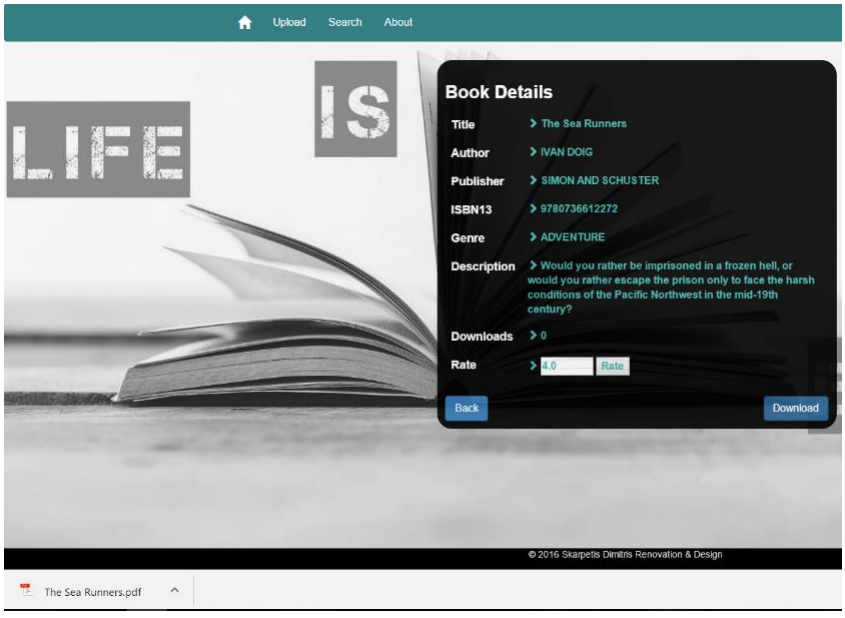
Πατώντας στο link “Details”, ο χρήστης οδηγείται στη παρακάτω σελίδα (εικόνα 6.10).



Εικόνα 6.10: Αναζήτηση βάση κατηγορίας.

Στο χρήστη εμφανίζονται αναλυτικά οι πληροφορίες του συγκεκριμένου βιβλίου, όπως για παράδειγμα το “Downloads” που φανερώνει τον αριθμό χρηστών που έχουν κατεβάσει αυτό το αρχείο. Στο πεδίο “Rate” ο χρήστης έχει τη δυνατότητα να βαθμολόγηση ένα βιβλίο. Κάθε φορά που ο χρήστης πατά το κουμπί “Rate”, αυτομάτως με χρήση Ajax τεχνολογίας, στο πεδίο εμφανίζεται ο μέσος όρος όλων των χρηστών.

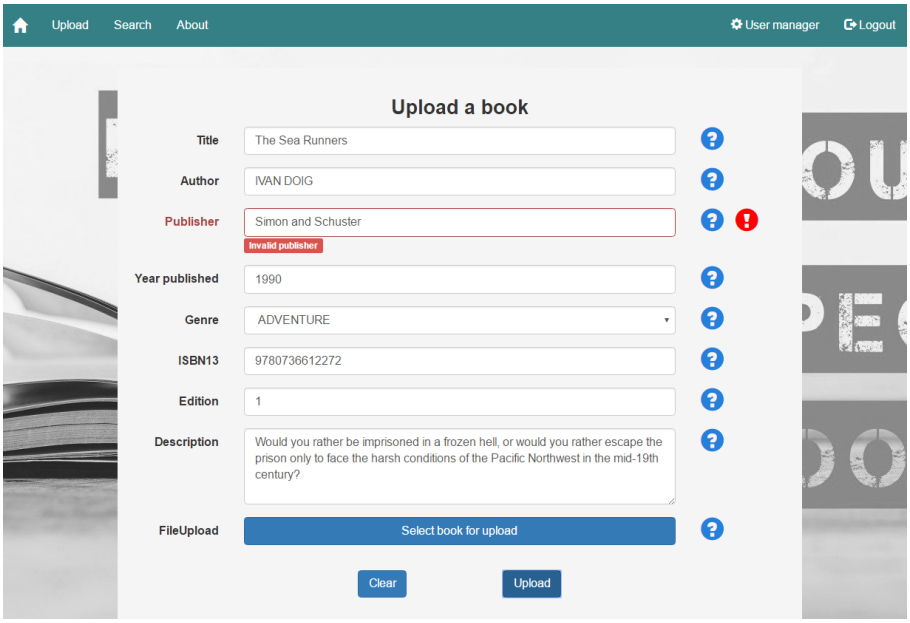
Τέλος ο χρήστης πατώντας το κουμπί “Download”, έχει τη δυνατότητα να αποθηκεύσει το αρχείο στο τοπικό δίσκο του υπολογιστή του (εικόνα 6.11).



Εικόνα 6.11: Download βιβλίου.

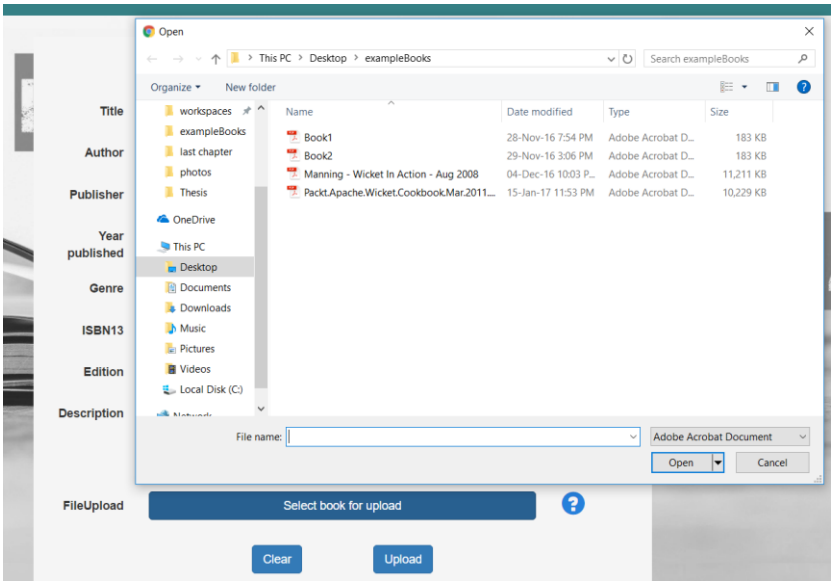
6.5 Upload βιβλίου

Ο χρήστης πατώντας το “Upload” link στο navigation bar της εφαρμογής μεταφέρεται στη σελίδα όπου μπορεί να ανεβάσει ένα ψηφιοποιημένο βιβλίο (εικόνα 6.12).



Εικόνα 6.12: Upload pdf βιβλίου.

Αρχικά συμπληρώνει όλα τα στοιχεία της φόρμας και στη συνέχεια πατώντας το κουμπί “Select book for upload” ανοίγει ένα παράθυρο πλοήγησης στο οποίο ο χρήστης θα επιλέξει το αρχείο για καταχώρηση στη βάση δεδομένων της εφαρμογής (εικόνα 6.13).

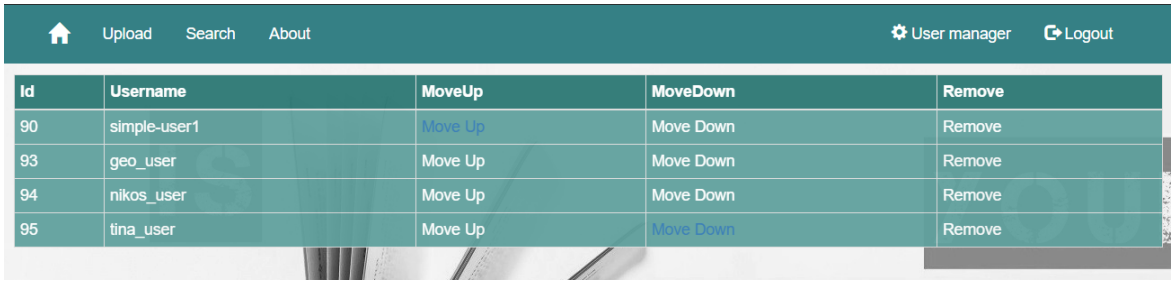


Εικόνα 6.13: Παράθυρο περιήγησης για καταχώρηση βιβλίου.

Το μέγιστο μέγεθος αρχείου που μπορεί ο χρήστης να καταχωρήσει στην εφαρμογή δεν πρέπει να ξεπερνά τα 40 megabytes, ενα όριο που υπάρχει μόνο για λόγους επίδειξης. Το συγκεκριμένο μέγεθος είναι μία παράμετρος της εφαρμογής, η οποία μπορεί να αλλάξει. Η διαδικασία ολοκληρώνεται με το πάτημα του κουμπιού “Upload”.

6.6 Διαχείριση χρηστών

Σε περίπτωση που ο διαχειριστής συνδεθεί στην εφαρμογή, τότε εμφανίζεται ο σύνδεσμος "User manager" μέσω του οποίου μεταφέρεται στη σελίδα διαχείρισης χρηστών (εικόνα 6.14).



Εικόνα 6.14: Σελίδα διαχείρισης χρηστών εφαρμογής.

Σε αυτή τη σελίδα εμφανίζονται όλοι οι εγγεγραμμένοι στην εφαρμογή χρήστες. Πατώντας το link “Remove” ο διαχειριστής έχει τη δυνατότητα να διαγράψει από τη βάση δεδομένων οποιονδήποτε απλό χρήστη.

Κεφάλαιο 7
Συμπεράσματα

7.1 Γενικά συμπεράσματα

Το Apache Wicket είναι ένα εργαλείο κατασκευής διαδικτυακών εφαρμογών στο οποίο κεντρικό ρόλο διαδραματίζουν τα components. Οποιοσδήποτε είναι εξοικειωμένος με τη γλώσσα προγραμματισμού Java, μπορεί άμεσα μετά από λίγες μέρες εκμάθησης του framework να γράψει ένα component. Το πόσο καλογραμμένο θα είναι εξαρτάται αποκλειστικά από την εμπειρία του προγραμματιστή και το επίπεδο εκμάθησης του framework.

Το μειονέκτημα του Wicket είναι ότι δεν παρέχει components με εξειδικευμένες λειτουργίες και μοντέρνα εμφάνιση. Το αδύνατο αυτό σημείο του αυτομάτως αποτελεί και το μεγαλύτερο πλεονέκτημά του. Το Apache Wicket παρέχει στον προγραμματιστή την ευελιξία και τον απόλυτο έλεγχο σε κάθε κομμάτι της εφαρμογής του. Μπορεί να δημιουργήσει ένα component ακριβώς όπως αυτός το έχει φανταστεί, τροποποιώντας τον κώδικα των HTML και Java αρχείων ή επιλέγοντας το κατάλληλο JavaScript framework. Ο προγραμματιστής κατασκευάζοντας το component της αρεσκείας του μπορεί να το επαναχρησιμοποιήσει οπουδήποτε μέσα στην ίδια ή σε διαφορετική εφαρμογή, να επεκτείνει τη λειτουργικότητα του και να το ενσωματώσει σε κάποιο άλλο.

Η πλειοψηφία των web frameworks βασίζεται στη χρήση “server side” κώδικα μέσα στα HTML αρχεία. Το γεγονός αυτό αντιβαίνει πλήρως την αρχή του διαχωρισμού της εμφάνισης από το περιεχόμενο της εφαρμογής (separation between presentation and business logic), πάνω στην οποία βασίστηκε και χτίστηκε το Apache Wicket framework. Στο Wicket το μόνο ειδικό tag που χρειάζεται να χρησιμοποιηθεί, είναι ένα αναγνωριστικό το οποίο προσθέτει το component (Java instance) στον HTML κώδικα. Εκτός του ότι ο κώδικας που παράγεται είναι πιο ευανάγνωστος και κατανοητός, ένα άλλο πλεονέκτημα ειδικά για μια εταιρεία, είναι ότι μπορεί να επιλέξει άτομα να εργαστούν είτε στο back end είτε στο front end ανάλογα με την εξειδίκευσή τους.

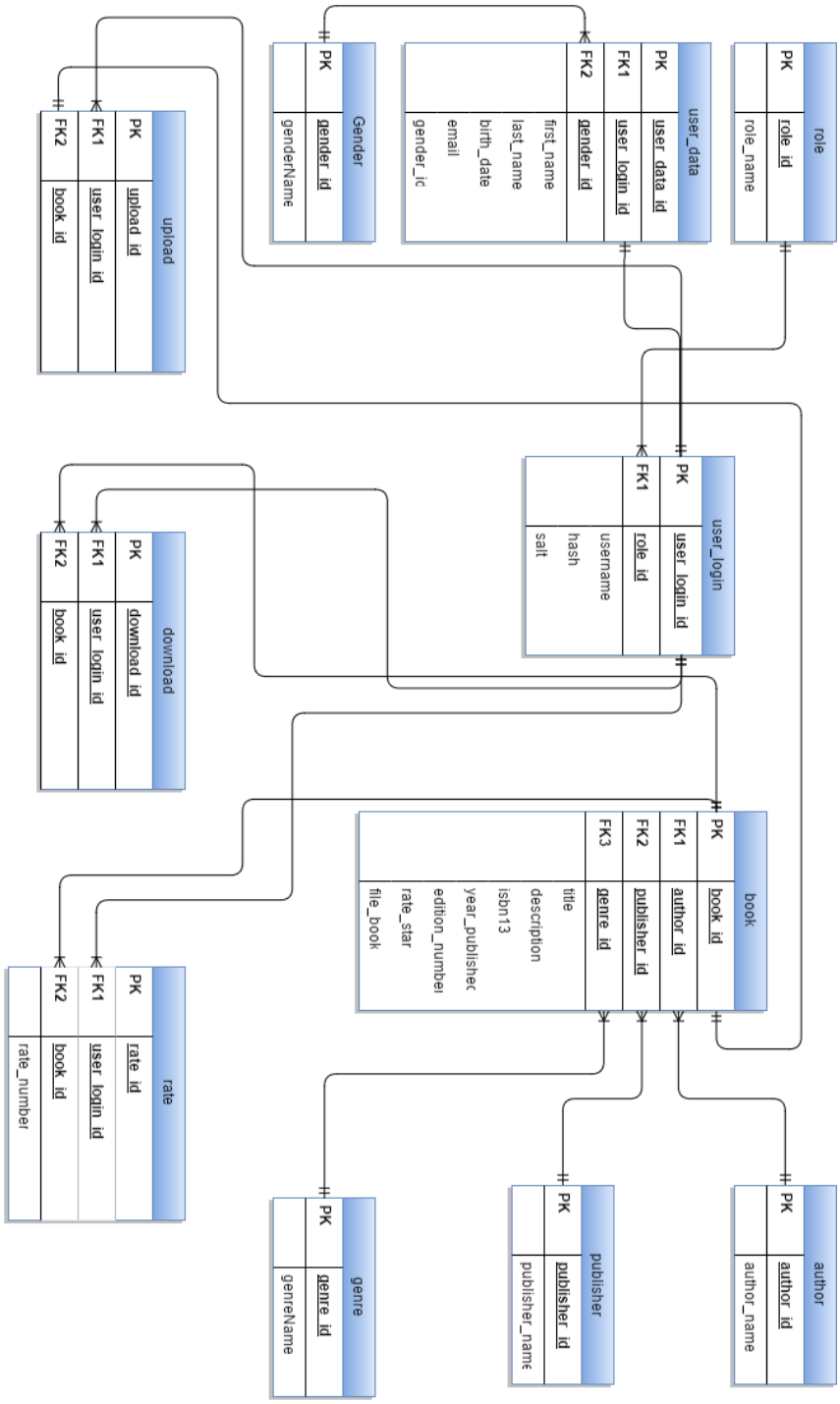
Το Apache Wicket είναι ένα framework με απεριόριστες δυνατότητες, που σου δίνει τη δυνατότητα να δημιουργήσεις υψηλής ποιότητας, ασφαλείς, συντηρήσιμες και επεκτάσιμες εφαρμογές. Απαραίτητη προϋπόθεση για να δουλέψει κάποιος με το συγκεκριμένο εργαλείο είναι το επίπεδό του στην γλώσσα προγραμματισμού

Java να είναι αρκετά προχωρημένο. Είναι αρκετά δύσκολο για έναν αρχάριο να προσθέσει λειτουργικότητα σε κάποιο από τα βασικά component που παρέχονται από το framework αν δεν είναι εξοικειωμένος με τις αρχές του αντικειμενοστραφούς προγραμματισμού.

ΠΑΡΑΡΤΗΜΑ Α’

Σχεσιακό μοντέλο βάσης δεδομένων

Το σχεσιακό μοντέλο της βάσης δεδομένων για την εφαρμογή ηλεκτρονικής βιβλιοθήκης.



ΒΙΒΛΙΟΓΡΑΦΙΑ

- Martijn Dashorst and Eelco Hillenius, “Wicket in Action”, Manning, August 2008 -ISBN: 9781932394986.
- Igor Vaynberg, “Apache Wicket Cookbook”, Packt Publishing, March 2011- ISBN: 978-1849511605.
- Karthik Gurumurthy, “Pro Wicket (Expert's Voice in Java)”, Apress - 1 edition, September 2006- ISBN: 978-1590597224.
- Joshua Bloch, “Effective Java”, Addison Wesley - 2 edition, May 2008 - ISBN: 978-0321356680.
- http://wpshopmart.com/best-java-web-development-frameworks/
- https://wicket.apache.org/learn/examples/markupinheritance.html
- https://cwiki.apache.org/confluence/display/WICKET/Component+hierarchy
- https://jaxenter.com/tutorial-apache-wicket-the-fun-web-framework-104503.html
- https://wicket.wordpress.com/2010/01/08/template-for-building- authenticated-webapplication/
- http://getbootstrap.com/
- https://www.abeautifulsite.net/whipping-file-inputs-into-shape-with-bootstrap-3
- https://www.postgresql.org/
- http://hibernate.org/
- https://www.tutorialspoint.com/hibernate/hibernate_query_language.htm
- http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html
- https://www.tutorialspoint.com/jpa/jpa_orm_components.htm
- http://www.journaldev.com/2403/google-guice-dependency-injection-example-tutorial
- http://www.journaldev.com/2882/hibernate-tutorial-for-beginners